École doctorale Informatique et Information pour la Société
Master Recherche Informatique de Lyon
Spécialité ECD (Extraction des Connaissances à partir des Données)

Rapport de stage

Algèbre XOLAP : analyse en ligne multidimensionnelle des entrepôts de données XML

Préparé au sein du Laboratoire ERIC - Université Lumière Lyon 2

Réalisé par

Marouane HACHICHA
marouane.hachicha@univ-lyon2.fr

Sous la direction de

M. Jérôme DARMONT
jerome.darmont@univ-lyon2.fr

M. Hadj MAHBOUBI
hadj.mahboubi@eric.univ-lyon2.fr

2006-2007

Résumé

Avec l'avènement de XML comme standard de représentation de données décisionnelles, les entrepôts de données XML trouvent leur place dans le développement de solutions décisionnelles basées sur le Web. Dans ce contexte, il devient nécessaire de permettre des analyses OLAP sur des cubes de données XML. Pour cela, des extensions du langage XQuery sont nécessaires. Afin de définir un cadre formel et de permettre l'optimisation indispensable des requêtes décisionnelles exprimées en XQuery, la définition d'une algèbre est souhaitable. Or, les algèbres XML-OLAP proposées dans la littérature dépendent encore largement du modèle relationnel et ne présentent qu'un nombre d'opérateurs OLAP très limité. En conséquence, nous proposons dans ce travail une nouvelle algèbre XOLAP (XML-OLAP) complète, issue de l'extension de l'algèbre XML TAX par des opérateurs OLAP. Nous mettons cette contribution en pratique grâce à une interface permettant d'exprimer les opérateurs de notre algèbre au sein du système de gestion de bases de données natif XML TIMBER.

Abstract

With the rise of XML as a standard for representing business data, XML data warehouses appear as suitable solutions for Web-based decision-support applications. In this context, it is necessary to allow OLAP analyses on XML data cubes. Thus, XQuery extensions are needed. To help define a formal framework and allow much-needed performance optimizations on analytical queries expressed in XQuery, defining an algebra is desirable. However, XML-OLAP algebras from the literature still largely rely on the relational model and only feature a very small number of OLAP operators. Hence, we propose in this work a new, complete XOLAP (XML-OLAP) algebra that is an extension of the TAX XML algebra by OLAP operators. We have also implemented this proposal through an interface that help express our algebra's operators within the TIMBER XML-native database management system.

Table des matières

1 Introduction 4

2 État de l'art 5

2.1 Algèbres OLAP 5

2.1.1 Extensions des concepts relationnels 5
2.1.1.1 Algèbre pour les cubes de données 5
2.1.1.2 Algèbres pour les cubes de données relationnels 5
2.1.1.3 Algèbre pour les cubes vus comme des ensembles de relations 6
2.1.1.4 Algèbres pour l'analyse en ligne OLAP 6
2.1.2 Extensions des concepts multidimensionnels 7
2.1.2.1 Algèbre pour les matrices bidimensionnelles 7
2.1.2.2 Algèbre pour les informations réparties en niveaux 7
2.1.2.3 Algèbre pour les tables multidimensionnelles 7
2.1.3 Discussion 8
2.1.3.1 Modèle de données 8
2.1.3.2 Opérateurs 8
2.1.3.3 Granularité des données 8

2.2 Algèbres XML 9

2.2.1 Algèbres pour les langages de requêtes XML 9
2.2.1.1 Algèbre pour les requêtes XML 9
2.2.1.2 Algèbre XAT 10
2.2.1.3 Algèbre XML pour XQuery 10
2.2.2 Algèbres sur les arbres de données XML 11
2.2.2.1 Algèbre pour la recherche d'information dans des fragments XML 11
2.2.2.2 Algèbre logique TAX et algèbre physique dérivée 11
2.2.2.3 Algèbre XAL 12
2.2.2.4 Algèbre Niagara 13
2.2.3 Algèbres sur les arbres de données XML pour l'évaluation de XQuery 13
2.2.3.1 Algèbre TLC 13
2.2.3.2 Algèbre de requêtes pour les flux de données XML fragmentées 14
2.2.4 Algèbres XML pour l'ECD 14
2.2.5 Discussion 14
2.2.5.1 Modèle de données 15
2.2.5.2 Unité de base de l'information 15
2.2.5.3 Opérateurs 15
2.2.5.4 Expressivité 16
2.2.5.5 Règles d'optimisation 16

2.3 Algèbres XOLAP 16

3 Algèbre XOLAP proposée 18

3.1 Problématique et objectif du stage 18

3.2 Justification du choix de TAX 18
3.3 Algèbre XOLAP 18
3.3.1 Modèle de données 18
3.3.2 Opérateurs 19
3.3.2.1 Opérateurs liés à la structure 19
3.3.2.2 Opérateurs ensemblistes 21
3.3.2.3 Opérateurs liés à la granularité 23
3.4 Discussion 26
3.4.1 Règles d'optimisation dans notre algèbre 26
3.4.2 Conformité entre notre algèbre XOLAP et l'algèbre OLAP classique 26
3.5 Mise en pratique de notre algèbre XOLAP 27
3.5.1 Recours à TIMBER 27
3.5.2 Mise en pratique de notre algèbre 27
3.5.2.1 Principe de l'application 27
3.5.2.2 Opérateurs implémentés 28
3.5.2.3 Opérateurs en cours d'implémentation 29
3.5.2.4 Limitations de XQuery rencontrées 29
4 Conclusion 30
Bibliographie 31
Annexes 36

1 Introduction

Les technologies décisionnelles telles que les entrepôts de données et l'analyse en ligne (OLAP, *On-Line Analytical Processing*) sont désormais technologiquement matures. Cependant, leur complexité les rend peu attractives pour de nombreux utilisateurs potentiels et les éditeurs de solutions décisionnelles commencent à développer des interfaces Web simples et conviviales [Law06]. De plus, de nombreuses applications décisionnelles nécessitent des sources de données externes à l'organisme qui les exploite. Par exemple, mener une veille concurrentielle au sein d'une entreprise requiert l'analyse de données uniquement disponibles auprès de ses concurrentes. Dans ce contexte, le Web est une source de données extrêmement riche. On parle notamment de *Web farming* [Hac99]. En conséquence, une nouvelle tendance à l'entreposage de données en ligne se dégage, avec des approches telles que l'entreposage virtuel [BCH99] ou l'entreposage XML [Pok02, BB03, HBH03, VBR03, PHS05, ZWLZ05, BMCA06].

Le langage XML (*eXtensible Markup Language* [Qui06]) est en effet de plus en plus utilisé comme standard pour représenter des données décisionnelles [BCC+05] et se montre particulièrement adapté pour modéliser des données dites complexes [DBRA05] issues de sources hétérogènes et notamment du Web. Ainsi, plusieurs travaux visent à étendre le langage XQuery [BCF+07] pour supporter des requêtes de type OLAP (groupement, agrégation, etc.) [BC04, BCC+05, Kay06, Ver06]. Ces extensions devraient non seulement permettre d'effectuer des analyses OLAP classiques, mais aussi de prendre en compte dans l'analyse en ligne des spécificités des données XML, comme par exemple des hiérarchies multiples, imbriquées et incomplètes (*ragged hierarchies* [BCC+05]), qui seraient très difficiles à gérer dans un environnement relationnel. Nous travaillons dans ce contexte à concevoir une algèbre XML-OLAP (ou XOLAP) permettant d'exécuter des requêtes OLAP sur des données XML natives. Notre objectif pour développer un tel outil est triple :

1. définir un cadre formel actuellement inexistant dans le contexte XOLAP ;
2. soutenir les efforts les efforts visant à étendre le langage XQuery pour permettre des analyses OLAP, notamment avec des opérateurs spécifiques à XML ;
3. permettre l'optimisation de requêtes OLAP exprimées en XQuery. Les Systèmes de Gestion de Bases de Données (SGBD) natifs XML, bien qu'en constant progrès, présentent en effet des limitations en terme de performance et bénéficieraient grandement d'une optimisation automatique des requêtes, et particulièrement des requêtes décisionnelles qui sont en général très coûteuses.

Ce mémoire est organisé comme suit. Le chapitre 2 détaille sur l'état de l'art sur les algèbres OLAP au début, les algèbres d'interrogation de données XML et les algèbres XOLAP, peu nombreuses, qui visent à permettre des analyses OLAP sur des cubes XML (algèbres XOLAP). Nous présentons dans le chapitre 3 notre propre algèbre XOLAP, qui se résume dans la mise en œuvre des opérateurs OLAP classiques dans un contexte XML. Pour cela, nous étendons l'algèbre XML TAX par des opérateurs OLAP. Nous mettons ensuite cette extension en pratique via une application générant des requêtes XQuery décisionnelles traduisant nos opérateurs. Ces requêtes seront par la suite exécutées dans le SGBD natif XML TIMBER. Finalement, nous présentons le bilan de nos travaux ainsi que ses perspectives dans le chapitre 4.

2 État de l'art

Malgré le développement des systèmes décisionnels, les bases de données multidimensionnelles et la technologie OLAP souffrent d'un manque de formalisation précise. Nous nous intéressons dans ce chapitre aux travaux qui proposent des modèles pour les données multidimensionnelles supportant une algèbre OLAP. Nous les classons en deux familles, selon les analogies qu'ils présentent avec les concepts relationnels et multidimensionnels, respectivement.

2.1 Algèbres OLAP

2.1.1 Extensions des concepts relationnels

2.1.1.1 Algèbre pour les cubes de données

Agrawal et al. proposent un modèle pour les bases de données multidimensionnelles et les opérateurs algébriques liés [AGS96, AGS97]. Ils modélisent les cubes multidimensionnels sous forme de repères à trois dimensions et proposent sur cette base une algèbre riche d'opérateurs inspirés essentiellement du relationnel, dont la sortie représente un nouveau cube. Conçus pour être traduits en SQL, ces opérateurs peuvent être mis en œuvre via un SGBD relationnel ou via un autre moteur supportant les données multidimensionnelles.

L'opérateur *push* transforme une dimension en mesure. L'opération réciproque est assurée par *pull*. L'opérateur de *destruction d'une dimension* supprime une dimension d'un cube, à condition que cette dernière ne dispose que d'une seule mesure. La *restriction* élimine d'une dimension les attributs qui ne satisfont pas un prédicat prédéfini. Le principe relationnel de *jointure* est transcrit en reliant deux cubes via leurs dimensions communes, tout en respectant une condition de jointure. L'opérateur *fusion* (*merge*) permet de réunir deux dimensions d'un cube en une seule et de calculer les mesures résultantes par l'intermédiaire d'une fonction d'agrégation.

Les auteurs définissent ensuite des opérateurs OLAP basés sur cette algèbre. Une *projection* sur un cube s'effectue en fusionnant les dimensions non concernées par cette opération en une seule, puis en la détruisant. L'*union* est assurée en joignant les valeurs non nulles de deux cubes ayant le même nombre de dimensions et utilisant les mêmes types de données. L'*intersection* s'effectue en joignant deux cubes dans les valeurs communes des dimensions. La *différence* entre deux cubes `C1` et `C2` est assurée par une *intersection* entre eux suivie d'une *union* du résultat avec le premier cube `C1`.

2.1.1.2 Algèbres pour les cubes de données relationnels

Li et Wang formalisent un cube de données relationnel pour une analyse OLAP [LW96]. Deux algèbres sont proposées dans ce contexte : une algèbre de groupement, extension de l'algèbre relationnelle, et une algèbre multidimensionnelle pour manipuler les cubes multidimensionnels.

Nous ne nous intéressons dans ce qui suit qu'à la deuxième. Dans le modèle de données correspondant, une dimension d'un cube est définie par son nom et une relation qui lui est associée. Le rôle des relations est d'assurer la correspondance entre les références des cellules et les attributs correspondants. Six opérateurs sont proposés, chacun fournissant un nouveau cube en sortie. Le premier opérateur, *ajout d'une dimension*, permet de générer un nouveau cube `C1` à partir d'un cube existant `C0`, en insérant une dimension dans ce dernier. Le *transfert* change l'emplacement d'un attribut dans une dimension, pour mieux visualiser les relations entre les attributs. L'*union* de deux cubes, selon une ou plusieurs dimensions de jointure, est possible s'ils disposent de la même structure. Plusieurs opérations d'agrégation sont possibles sans passer par des opérations de groupement. *L'agrégation de cubes* permet de réduire la taille d'un cube en compressant les mesures relatives à ses dimensions. L'opérateur *rc-join* permet de joindre une relation avec une dimension d'un cube. Finalement, la *construction* génère un cube à partir d'une relation. Les dimensions et les mesures concernées sont spécifiées en entrée de l'opérateur.

2.1.1.3 Algèbre pour les cubes vus comme des ensembles de relations

Avec pour objectif l'analyse en ligne OLAP, Gyssens et Lakshmanan modélisent les bases de données multidimensionnelles en tables relationnelles [GL97]. Cette représentation conserve les noms des tables de faits et de leurs attributs, pour les attributs dimensionnels comme pour les mesures.

Très proche de l'algèbre relationnelle, cette algèbre se compose de deux ensembles d'opérateurs. Le premier ensemble repose sur des *opérateurs classiques* repris de l'algèbre relationnelle (*union*, *intersection*, *différence* et *produit cartésien*) et les opérateurs unaires de *sélection*, de *projection* et de *renommage*. Chacun de ces opérateurs est défini de la même façon que dans le cas relationnel. Le second ensemble d'opérateurs propose deux *opérateurs de restructuration*, *unfold* et *fold*, qui permettent de remanier la structure d'un cube. *Unfold* ajoute une nouvelle dimension dans le schéma relationnel d'un cube. Réciproquement, l'élagage d'une dimension d'un cube est assuré par *fold*.

2.1.1.4 Algèbres pour l'analyse en ligne OLAP

Datta et Thomas proposent un modèle de cube de données supportant une algèbre OLAP [DT99]. Ce modèle de cube respecte la symétrie entre les dimensions et les mesures, comme la symétrie entre les agrégations de types *forage vers le haut* (*roll up*) et *valeurs numériques* (*somme*, *moyenne*, etc.), ou encore entre les transformations (*convertir une dimension en mesure* et *vice versa*). Cette symétrie est à l'origine des opérateurs OLAP proposés par les auteurs. Ces derniers définissent un cube comme un quadruplet < D, M, A, f >. D et M désignent les dimensions et les mesures, respectivement. Chaque D ou M est définie par un nom relatif à un domaine prédéfini. Les ensembles des noms des dimensions et des noms d'attributs sont disjoints. A représente un groupe d'attributs, chacun étant défini par son nom, extrait du domaine de noms d'attributs. Enfin, f représente une fonction de D vers A qui permet d'établir la correspondance entre les dimensions et les attributs. Les auteurs modélisent aussi un cube matérialisé dans l'ensemble < D, M, A, f, V, g >. V représente l'ensemble des valeurs utilisées dans le cube, tandis que g symbolise la combinaison des domaines des noms des dimensions associées à ces valeurs.

Sur ce modèle, l'algèbre des auteurs propose neuf opérateurs dédiés à la manipulation OLAP. Le premier opérateur, *restriction*, permet de limiter un ensemble des valeurs V pour obtenir en sortie un ensemble V0 inclut ou égal à V. L'opérateur *agrégation* permet d'agréger une mesure en groupant les attributs de la dimension correspondante. Le *produit cartésien* relie deux cubes en unifiant les dimensions d'une part et les mesures d'autre part. La *jointure* est un cas particulier du *produit cartésien*, conçue dans le but de relier deux cubes ayant une ou plusieurs dimensions communes. Les deux opérateurs suivants nécessitent que les deux cubes en question soient union-compatibles. L'*union* permet d'unir deux cubes. C'est une opération différente du *produit cartésien* puisqu'il s'agit d'unifier les ensembles de valeurs de deux cubes en une seule valeur V0 dans le cube résultant. La *différence* consiste à soustraire l'ensemble de valeurs correspondant au deuxième cube du premier. La *partition* classe les cellules d'un cube dans des groupes significatifs. Cette tâche concerne certains types d'agrégation. Par exemple, il est possible de grouper toutes les moyennes selon un prédicat prédéfini. Les deux derniers opérateurs, *pull* et *push*, peuvent être classés comme des opérateurs de transformation. *Pull* permet de convertir les mesures en dimensions selon une fonction de transformation. *Push* permet de réaliser l'opération inverse en utilisant aussi une autre fonction de transformation dédiée.

Un second travail des mêmes auteurs utilise ce modèle et propose une autre algèbre de neuf opérateurs [TD01]. Sept d'entre eux ne sont pas modifiés. La *restriction*, l'*agrégation*, l'*union* et la *différence* ont conservé les mêmes noms et principes. Le *produit cubique*, les *extractions des dimensions* et *des mesures* remplacent le *produit cartésien*, *pull* et *push*, respectivement. Deux nouveaux opérateurs sont la *projection métrique* et le *renommage*. La *projection métrique* réduit le nombre d'attributs correspondant aux mesures prises en compte dans la représentation des données. Le *renommage* permet de rebaptiser les éléments d'un ensemble d'attributs ou de caractéristiques (mesures ou dimensions).

2.1.2 Extensions des concepts multidimensionnels

2.1.2.1 Algèbre pour les matrices bidimensionnelles

Gyssens et al. proposent une algèbre pour les données multidimensionnelles modélisées de façon tabulaire [GLS96]. Rangées dans des tableaux bidimensionnels, les données peuvent être des noms, des valeurs numériques ou nulles. En l'absence d'une structure figée, les tables peuvent prendre plusieurs formes tout en respectant l'aspect multidimensionnel des données. Les auteurs définissent ensuite une algèbre tabulaire et établissent une catégorisation des opérateurs.

Les opérateurs traditionnels proposés sont l'*union*, la *différence*, le *produit cartésien*, le *renommage*, la *projection* et la *sélection*. Ces derniers fonctionnent de la même façon que leurs équivalents en algèbre relationnelle. D'autres opérateurs de restructuration sont proposés : le *groupement* (*grouping*), la *fusion* (*merging*), l'*éclatement* (*spliting*) et la *compression* (*collapsing*). Le *groupement* et la *fusion* (respectivement l'*éclatement* et la *compression*) peuvent être vus comme l'inverse l'un de l'autre. Le *groupement* permet de réunir, dans un tableau, les informations selon un certain attribut. La *fusion* effectue l'opération réciproque. L'*éclatement* permet de transformer une table en plusieurs. Par contre, la *compression* fusionne un ensemble de tables en un seul. Il est clair alors que *groupement* et *fusion* sont deux opérateurs qui manipulent essentiellement les données, alors qu'*éclatement* et *compression* manipulent leur structure. Grâce à l'opérateur de *transposition* (*transposing*), il est possible d'exprimer, pour toute opération sur les lignes d'une matrice, une opération similaire sur ses colonnes. Finalement, afin d'offrir une meilleure représentation des données, la *suppression des redondances* (*clean-up*) élimine dans une table les valeurs nulles.

2.1.2.2 Algèbre pour les informations réparties en niveaux

Cabibbo et Torlone proposent un modèle logique des systèmes OLAP pour la conception des bases de données multidimensionnelles [CT97, CT98]. Dans ce modèle, les dimensions sont représentées en hiérarchies, suivant un ordre prédéfini exprimé par une relation de forage vers le haut.

Une famille de fonctions, appelées *fonctions de forage vers le haut* (*roll up functions*), complète la définition d'une dimension. Le schéma d'un modèle multidimensionnel consiste alors en un ensemble fini de dimensions et un ensemble fini de tables de faits. Malgré l'intérêt de leur approche, Cabibbo et Torlone se limitent à la formalisation de cette famille de *fonctions de forage vers le haut* et ne proposent pas d'autres opérateurs.

2.1.2.3 Algèbre pour les tables multidimensionnelles

Ravat et al. proposent un modèle qui sert de support à une algèbre et à un langage graphique pour l'analyse et la visualisation des données [RTZ06a, RTZ06b]. Leur algèbre intègre les opérateurs OLAP classiques complétés par des opérateurs avancés afin de faciliter l'expression d'analyses décisionnelles complexes. Le modèle repose sur une représentation multi-faits où chaque fait est analysé en fonction d'axes d'analyse (dimensions) multi-vues (multi-hiérarchisées). Les auteurs modélisent un cube de données par une structure de visualisation proche des arbres d'attributs, sous la forme d'un tableau à double entrée hiérarchisé appelé table multidimensionnelle (TM) [RTZ01].

L'algèbre proposée repose sur un opérateur de *construction* (*display*) qui produit une TM à partir d'une base de données multidimensionnelle et sur un ensemble de onze *opérateurs fondamentaux* portant sur les TM et facilitant la manipulation OLAP. Le premier opérateur, la *rotation*, permet de remplacer un axe d'analyse par un autre, ou de changer la hiérarchie sur un même axe au sein d'une TM. Les *forages vers le bas* ou *vers le haut* (*drill down* ou *roll up*) autorisent la modification des différents niveaux de granularité utilisés pour visualiser les données au sein d'une TM. Les dimensions de la TM peuvent être enrichies par des attributs supplémentaires en employant l'opérateur d'*imbrication* (*nest*). La *sélection* restreint l'ensemble des valeurs affichées des attributs dimensionnels et des mesures correspondantes. L'opérateur d'*organisation* (*switch*) intervertit deux valeurs d'un attribut d'une dimension pour permettre l'ordonnancement des valeurs affichées. Le *calcul d'agrégats* permet d'ajouter dans une TM des calculs agrégeant ses lignes et/ou ses colonnes. L'opérateur de *conversion d'un paramètre en mesure* (*push*) transforme un paramètre (dimension) en

mesure au sein de la TM. La *conversion d'une mesure en paramètre* (*pull*) transforme une mesure en paramètre (dimension). Enfin, les opérateurs d'*ajout* (*addm*) et *de suppression* (*delm*) *de mesures* permettent de modifier l'ensemble des mesures calculées.

2.1.3 Discussion

Pour évaluer les différentes algèbres OLAP présentées dans les sections précédentes, nous avons déterminé trois critères de comparaison qui sont, généralement, transversaux aux familles d'algèbres que nous avons identifiées dans un premier temps (extensions des concepts relationnels d'une part et multidimensionnels d'autre part). Le modèle de données adopté, les opérateurs offerts et la gestion de la granularité des données constituent l'ensemble de ces critères.

2.1.3.1 Modèle de données

Dans les travaux que nous venons de présenter, chaque algèbre est basée sur un modèle de données spécifique. Bien que tous ces modèles partent du principe d'analyser des faits OLAP selon plusieurs dimensions et mesures, leur conception diffère d'une algèbre à l'autre.

Nous retrouvons ici les deux grandes familles de modèles supportées par les différentes algèbres. La première famille adopte une extension du modèle relationnel, où les auteurs modélisent les données multidimensionnelles dans un format relationnel classique [GLS96, LW96, GL97]. Au contraire, dans la deuxième famille, les concepts relationnels sont exclus de toute représentation des données [AGS96, AGS97, CT97, CT98, TD01, RTZ06a, RTZ06b]. Par exemple, dans leur travail, Agrawal et al. modélisent les données en un espace de représentation multidimensionnel sous forme d'axes de dimensions, sans se référer aux tableaux. Les autres auteurs ont aussi laissé de côté la représentation tabulaire des données, à l'exception de Cabibbo et Torlone, qui modélisent les données multidimensionnelles dans des tables (mais ces dernières sont loin d'être relationnelles).

Pour conclure, cette diversité de modèles provient du fait que les systèmes d'aide à la décision reposant sur la technologie OLAP ont existé avant la définition d'un fondement théorique standard et reconnu par la communauté des bases de données.

2.1.3.2 Opérateurs

Outre les opérateurs OLAP classiques - la *rotation* (*rotate*), la *permutation d'une modalité dans une dimension* (*switch*), le *forage vers le haut* ou *vers le bas* (*roll up* ou *drill down*), la *sélection* et la *projection* sur un cube (*slice* et *dice*), la *traduction des mesures en dimensions* ou *des dimensions en mesures* (*pull* ou *push*) et l'opérateur *cube* - et les opérateurs inspirés du relationnel, peu d'opérateurs ont été proposés pour enrichir l'analyse en ligne en général.

Parmi ceux-ci, nous pouvons cependant citer la *suppression des redondances* dans l'algèbre de Gyssens et al. ou encore les *opérateurs de restructuration* (*unfold* et *fold*) de Gyssens et Lakshmanan. L'inconvénient de ces nouveaux opérateurs est qu'ils sont totalement dépendants du modèle de données. Leur utilisation dans le cadre d'une autre algèbre OLAP nécessite une adaptation au modèle qui lui est associée. De plus, nous pouvons également noter que ces opérateurs ont des synonymes dans d'autres algèbres. Par exemple, *unfold* est équivalent à l'opérateur *add dimension* de Li et Wang, tandis que *fold* est équivalent à *destroy dimension* de Gyssens et al.

2.1.3.3 Granularité des données

La prise en compte de la granularité des données varie largement selon les algèbres OLAP. Certaines ne considèrent d'ailleurs pas du tout la granularité des données dans leurs différents opérateurs [GLS96, GL97], vraisemblablement du fait que ces algèbres s'inspirent (trop) directement du cas relationnel.

La prise en compte de la granularité des données est toujours accompagnée par une implémentation des opérateurs de *forage vers le haut* et/ou *vers le bas* (*roll up* et/ou *drill down*) et parfois de l'opérateur *cube*. Agrawal et al. expriment le *forage vers le haut* (*roll up*) en associant une fonction d'agrégation, comme *somme*, à l'opérateur *fusion* de leur algèbre. Ils expriment aussi le

forage vers le bas (*drill down*) en effectuant l'opération réciproque du *forage vers le haut*, c'est-à-dire en « remontant en arrière » pour savoir comment sont réalisées l'*agrégation* et la *fusion* des données. Li et Wang présentent, dans leur algèbre de groupement, un opérateur de *forage vers le haut* (*roll*) qui se repose sur une opération de *groupement*. Les *fonctions de forage vers le haut* de Cabibbo et Torlone permettent également d'atteindre ce but, mais les auteurs ne donnent malheureusement pas plus de détails sur leur fonctionnement. Bien que Datta et Thomas ne présentent pas d'opérateurs de *forage* ni d'*agrégation*, ils affirment que leur algèbre peut supporter ces opérations. Enfin, Ravat et al. développent des opérateurs de *forage vers le haut* et *vers le bas* (*roll up* et *drill down*) qui ne s'éloignent pas, dans leur fonctionnement, de leurs équivalents dans les autres algèbres.

2.2 Algèbres XML

Dans cette section, nous étudions les algèbres XML présentées dans la littérature. Bien que la plupart puisent leur logique dans celle des opérateurs relationnels, de nouveaux opérateurs sont apparus avec XML. Nous classons dans les sections suivantes les algèbres XML en quatre familles, selon la logique d'implémentation suivie :

1. algèbres pour les langages de requêtes XML ;
2. algèbres sur les arbres de données XML ;
3. algèbres sur les arbres de données XML pour l'évaluation de XQuery ;
4. algèbres XML pour l'Extraction des Connaissances à partir des Données (ECD).

2.2.1 Algèbres pour les langages de requêtes XML

2.2.1.1 Algèbre pour les requêtes XML

En s'inspirant essentiellement des notions issues de multiples langages comme SQL [DD97], XPath [CD99] et NRA (algèbre relationnelle d'imbrication [Col90, BNTW95, LMW96, LW97]), Fernández et al. ont soumis au W3C (*World Wide Web Consortium*) une algèbre pour les requêtes XML riche et facile à utiliser [FSW00]. Les documents et les schémas XML y sont transformés en nouveaux schémas adaptés à l'algèbre. Deux notions importantes les caractérisent : les types et les variables globales. Un type est un élément incluant d'autres éléments, comme la racine d'un document XML. Une variable globale représente une instance du document XML.

Cette algèbre comprend de nombreux opérateurs, dont le plus simple est la *projection*, puisqu'il suffit de préciser le chemin de l'élément recherché, de façon similaire à XPath. Le deuxième opérateur, l'*itération*, consiste à parcourir les éléments indiqués dans le chemin employé, via une boucle for. Itérer revient alors à associer une *projection* au for. Cette association constitue la base des quatre opérateurs suivants. Une *sélection* est possible en associant une clause where à une *itération*. L'opérateur suivant, la *quantification*, vise à sélectionner les éléments d'un type quelconque ayant des sous-éléments communs. Pour joindre des éléments d'un ou plusieurs documents (*jointure*), il suffit d'imbriquer des boucles for. Le nombre de boucles for utilisées correspond au nombre de sources (types) à joindre en un seul élément. Le résultat est donné selon l'ordre d'imbrication. Une clause where peut être employée pour conditionner cette opération. Le quatrième opérateur suivant cette logique, la *restructuration*, regroupe différents éléments dans un seul document XML selon une ou plusieurs conditions. Des clauses for et where sont employées afin d'atteindre cet objectif. De plus, la fonction distinct est utilisée pour éviter les redondances, comme dans le cas relationnel.

En plus de ces six opérateurs, les fonctions d'*agrégation* suivantes sont inclues dans l'algèbre : la *moyenne* (*avg*), le *comptage* (*count*), le *maximum* (*max*), le *minimum* (*min*) et la *somme* (*sum*). Elles sont toujours associées à une condition de type where et tiennent compte des répétitions de types. Finalement, une nouvelle notion spécifique à cette algèbre est introduite, les *fonctions*. Leur définition suit le même principe que dans les langages de programmation classiques (nom, paramètres et type de retour). Le but est de généraliser, via une fonction, une opération donnée. L'utilisateur peut définir un nombre illimité de fonctions et utiliser des fonctions récursives.

2.2.1.2 Algèbre XAT

Zhang et al. proposent une algèbre pour les langages de requêtes XML [ZPR02]. Baptisée XAT (*XML Algebra Tree*), elle permet de modéliser les requêtes XQuery sous forme d'arbres avant de les optimiser au sein du système Rainbow [ZDW+03]. Les auteurs soutiennent l'idée de traduire les données XML en relationnel, mais utilisent aussi ces deux paradigmes simultanément. En entrée de Rainbow, les données sont stockées dans des documents XML avant d'être transférées dans des SGBD relationnels.

XAT est essentiellement conçue pour améliorer les performances de Rainbow en optimisant les requêtes. Elle présente trois catégories d'opérateurs : opérateurs XML, opérateurs SQL et opérateurs spécifiques XAT. Les opérateurs classiques (*projection*, *sélection*, etc.) sont exprimés en SQL. Les opérateurs spécifiques assurent les différentes phases d'optimisation. Les opérateurs XML d'*union*, d'*intersection*, de *différence* et d'*agrégation* opèrent sur les colonnes où sont stockées les données XML après leur transformation en relationnel. L'*exposition* (*expose*) permet de visualiser ces colonnes comme des fragments ou des documents XML ; l'*étiquetage* (*tagger*) d'étiqueter une liste de données selon un modèle ; et enfin, la *navigation* (*navigate*) de parcourir les données XML contenues dans une colonne selon le chemin indiqué.

2.2.1.3 Algèbre XML pour XQuery

Novak et Zamulin proposent une algèbre pour XQuery qui se limite volontairement à des structures de premier ordre [NZ06]. Cette algèbre est basée sur des expressions et respecte un modèle (*signature*) de base de données XML spécifié par les auteurs.

Une expression élémentaire est une séquence de nœuds (*sequel*) de l'arbre qui représente le document XML. Elle est notée *Seq(Node)*→*Seq(Node)*.Cette séquence peut respecter l'ordre des nœuds dans l'arbre (*document_order*) ou non (*reverse_order*). Une expression incorpore aussi des accesseurs de nœuds (*node-kind*, *node-name*, *parent*, *string-value*, *type*, *children*, *attributes* et *nilled*).

Afin d'orienter le parcours de ces expressions dans l'arbre XML, les auteurs définissent et formalisent des fonctions de navigation (parcours). Ces fonctions représentent les différents axes d'un arbre XML et permettent de pointer un ou plusieurs nœuds spécifiques (*child*, *descendent*, *descendent_or_self*, *parent*, *ancestor*, *ancestor_or_self*, *self*, *attribute*...). Une expression élémentaire qui adopte une fonction de navigation est notée *function_n*ame : *Node* ⟶ *Seq(Node)*.

Une expression FLWOR du langage XQuery peut ainsi être définie par la combinaison de plusieurs expressions élémentaires. Les *expressions de parcours* (*path expressions*) sont exprimées dans les clauses for et let, les *expressions de sélection* (*selection expressions*) dans la clause where, *les expressions d'ordre* (*ordering expressions*) dans la clause order by et les e*xpressions de construction du résultat* (*mapping expressions*) dans la clause return.

Une *sélection* (*selection expression*) consiste à extraire une sous-séquence (autre expression) en se basant sur un critère de sélection. Elle est notée *s :: p*, où *s* représente une expression (*sequence*) et *p* la condition de sélection. Cette condition peut être exprimée par l'évaluation d'un prédicat (*predicate test*) ou par un test sur les nœuds pointés par l'expression (*kind test*). Cette fonctionnalité est spécifique à XML et n'existe pas dans les modèles relationnels et orientés-objet. Une *projection* est définie par des expressions orientées par des fonctions de navigation. Elle consiste à sélectionner différentes parties de l'arbre XML. Une *jointure* revient à des expressions de « désemboîtement » (*unnesting join expression*) dues à la nature complexe des arbres XML. En effet, la jointure est définie par des relations entre identifiants (type de nœud, contenu...) de séquences de nœuds et non par des clés (comme en relationnel) et la position des séquences dans l'arbre. L'évaluation d'une jointure revient donc à comparer des sous-arbres. Finalement, les auteurs définissent de nouvelles expressions qui permettent la construction de nœuds (*document*, *element*, *attribute and text node constructors*). Ces expressions permettent d'ajouter de nouveaux nœuds à l'arbre initial ou de créer un nouvel arbre (ou séquence).

2.2.2 Algèbres sur les arbres de données XML

La représentation des documents XML sous forme d'arbres de données enracinés et ordonnés (plus rarement non ordonnés) est défendue par de nombreux auteurs. Dans cette représentation, la racine du document devient le nœud racine de l'arbre et ses éléments fils deviennent les nœuds fils de cette racine, et ainsi de suite pour le reste des éléments.

2.2.2.1 Algèbre pour la recherche d'information dans des fragments XML

Selon Pradhan, trouver un fragment intéressant dans un document XML revient à identifier le sous-arbre correspondant [Pra06]. Comme il est possible de parvenir à n'importe quel nœud en suivant différents chemins, le fragment recherché doit être composé d'un ensemble de nœuds reliés ensemble par le plus proche nœud parent commun.

Pradhan propose une algèbre composée de deux opérateurs : la *sélection* et la *jointure*. En définissant un prédicat (contrainte), une *sélection* consiste à identifier tous les fragments satisfaisant ce prédicat. Une *jointure* de deux fragments f1 et f2 est possible si et seulement s'ils contiennent ou sont égaux à la condition de jointure, un fragment f. De plus, il ne faut pas qu'il existe un autre fragment f' inclus et plus réduit que f, f1 et f2 en même temps. En d'autres termes, la *jointure* se réalise sur le plus petit fragment commun entre les sous-arbres à joindre. La *jointure* entre fragments utilise les propriétés algébriques suivantes : l'idempotence, la commutativité, l'associativité et l'absorption. L'algèbre offre aussi la possibilité de joindre un nombre infini de paires d'ensembles de fragments F1 et F2. L'opération consiste à appliquer le principe décrit ci-dessus pour toute combinaison de paires possible. Dans ce cas, l'auteur utilise les propriétés algébriques de commutativité, d'associativité, de monotonie et de loi de distribution. Cette *plus puissante jointure de fragments* (*powerset fragment join*) représente une extension de l'opérateur de *jointure* dans laquelle un nombre arbitraire non nul de fragments de F1 et F2 sont joints.

2.2.2.2 Algèbre logique TAX et algèbre physique dérivée

Jagadish et al. proposent une algèbre logique d'arbres nommée TAX (*Tree Algebra for XML*), qui est une extension de l'algèbre relationnelle [JLST01]. Pour définir les arbres XML, les auteurs s'inspirent de DOM (*Document Object Model*) [HWW06] et modélisent chaque nœud par une paire attribut-valeur. À partir d'une racine unique, chaque occurrence d'un élément fils représente un sous-arbre séparé. La descente dans la hiérarchie utilise le même principe. Jagadish et al. introduisent également des modèles d'arbres (*pattern trees*) que doivent suivre les arbres en sortie des requêtes, dénommés arbres témoins (*witness trees*).

Dans TAX, la *sélection* retourne un arbre témoin satisfaisant le prédicat indiqué dans le modèle d'arbre. En entrée de cet opérateur, nous trouvons l'arbre modélisant la base, le modèle et une liste de paramètres. La sortie représente la correspondance entre la base et le modèle, en tenant compte des éléments en paramètres. La *projection* est similaire à la *sélection* dans le sens qu'elle retourne les nœuds non spécifiés dans l'ensemble d'entrée. Bien qu'ils fonctionnent sur le même principe, *projection* et *sélection* sont deux opérateurs complètement différents, indépendants et surtout non-orthogonaux, au contraire du cas relationnel (les arbres n'ont pas une structure faite de lignes et de colonnes). Le *produit* prend en entrée deux collections d'arbres, C et D, pour produire en sortie une juxtaposition des différentes paires de fragments possibles. Comme dans le cas relationnel, l'ordre est très important ici car C*D est différent de D*C. De plus, une *jointure* peut être assurée par un *produit* suivi d'une *sélection*.

Les opérateurs ensemblistes (*union*, *intersection* et *différence*) ne peuvent être utilisés dans TAX qu'en cas de similitude entre deux arbres. Un certain isomorphisme doit exister entre les nœuds des arbres traités afin de conserver leur ordre ainsi que les paires attribut-valeur correspondantes. Le *groupement* n'est pas ignoré dans ce travail mais, contrairement au cas relationnel, il est séparé des opérations d'*agrégation*. Pour grouper des nœuds, *group by* prend en entrée un modèle d'arbre, une fonction de groupement pour identifier les arbres témoins dans la collection initiale selon ce modèle, et une fonction d'ordre pour une meilleure organisation du résultat. En sortie du *groupement*, les doublons sont éliminés. Les opérateurs classiques d'*agrégation* comme la *somme*, le *minimum*, le

maximum et le *comptage* sont spécifiés dans TAX. Ils prennent en entrée un modèle d'arbre, la nature de l'agrégation à réaliser et une spécification de mise à jour. Cette dernière précise comment est calculée l'agrégation et comment elle est insérée dans l'arbre initial pour aboutir à l'arbre témoin, selon le modèle d'arbre en entrée.

D'autres opérateurs TAX n'ont pas d'équivalent dans les travaux similaires. Leur but principal est de mettre à jour les arbres. Le premier, le *renommage* (*renaming*), consiste à changer le nom ou la valeur des attributs associés à des nœuds spécifiques de l'arbre. La liste de noms possibles est définie en entrée de cet opérateur. Le *renommage* TAX est analogue au *renommage* des attributs dans les tables relationnelles. L'opérateur de *réordonnancement* (*reordering*) permet de modifier l'ordre des descendants d'un nœud donné, ou encore le groupement de plusieurs nœuds en un seul. Le *réordonnancement* prend en entrée une collection d'arbres, le modèle à suivre et la liste des nœuds cibles. Les auteurs ont également enrichi leur travail par un opérateur *copier-et-coller* (*copy-and-paste*) qui permet de dupliquer de nouveaux nœuds dans l'arbre. Une autre opération de mise à jour des arbres est également proposée via la *mise à jour des valeurs* (*value updates*) des attributs des nœuds. Ici, les nœuds cibles sont définis dans le modèle. Une liste spécifiant les noms possibles, qui peuvent être constants, dérivés d'autres noms ou encore déjà existants, est associée au modèle en entrée de l'opération. L'algèbre est également munie d'un opérateur de *suppression de nœuds* (*node deletion*), pour lequel une liste de nœuds à supprimer doit être définie. Le dernier opérateur est l'*insertion de nœuds* (*node insertion*). Lors de cette opération, certaines informations doivent être fournies : la position d'insertion ainsi que l'adaptabilité des nœuds associés à l'objet inséré. Le *copier-et-coller* peut être utilisé pour indiquer les nœuds représentant les futurs descendants du nœud inséré.

2.2.2.3 Algèbre XAL

Frasincar et al., auteurs de l'algèbre XAL (*XML Algebra*), adoptent également la représentation des documents XML sous forme d'arbres et plus généralement de graphes [FHP02]. XAL comprend trois familles d'opérateurs.

Les opérateurs d'extraction recherchent l'information nécessaire dans les graphes XML et renvoient la collection de sommets correspondante. La *projection*, la *sélection*, la *distinction* (*distinct*), la *jointure*, le *désordonnancement* (*unorder*) et le *tri* (*sort*) constituent l'ensemble de ces opérateurs. Ils s'appliquent sur des graphes. Les quatre premiers ne présentent pas de différence avec les opérateurs TAX. En revanche, le *désordonnancement* permet de désorganiser une collection en entrée ; tandis que le tri transforme une collection suivant un modèle donné.

Les méta-opérateurs contrôlent l'évaluation des expressions. Ce ne sont pas de véritables opérateurs au sens de la création ou de l'extraction d'éléments de données. Ils sont plutôt utilisés pour exprimer des répétitions au niveau des opérateurs ou des entrées. *Map* applique une fonction prédéfinie sur chaque élément de la collection d'entrée. L'effet de cette fonction se révèle dans la collection de sortie. *Kleene Star* répète une fonction, indéfiniment, pour une collection d'entrée donnée. À chaque itération, le résultat est ajouté en entrée de la fonction suivante. Par rapport à *Map*, *Kleene Star* inclut la collection d'entrée en plus dans le résultat.

Finalement, les opérateurs de construction permettent de manipuler la structure des documents XML (nouveaux sommets et arêtes). La *création de sommet* (*create vertex*), premier opérateur de construction présenté, ajoute un nouveau sommet (nœud) au graphe. Le type et la valeur (même nulle) des nœuds sont indiqués en entrée de l'opérateur. L'ajout de nouvelles arêtes est assuré via l'opérateur de *création d'arête* (*create edge*), qui prend en entrée les nœuds père et fils à relier, ainsi que le type et le nom de la nouvelle liaison, et renvoie le sommet fils. Pour terminer, partant du besoin de copier des sommets et des arêtes dans le but d'enrichir les graphes, Frasincar et al. définissent un opérateur de *copie d'exemples* (*copy examples*). Copier un sommet revient à créer un nouveau sommet avec le type et la valeur du sommet original. Copier une arête nécessite la copie de tous les nœuds, parents et fils, correspondants. En conséquence, copier un graphe complet, en partant d'un sommet donné, consiste à copier tous ses sous-graphes ainsi que les graphes les reliant à ce sommet.

2.2.2.4 Algèbre Niagara

Niagara est une algèbre pour langages de requêtes XML qui se base également sur une modélisation des documents XML en graphes [GVD+01]. Dans ce modèle, les attributs sont modélisés au niveau le plus bas de l'arbre.

Le premier opérateur de Niagara, *source*, prend en entrée un ou plusieurs types de documents (XML, DTD, XML+DTD). En sortie, il fournit une collection d'ensembles de singletons représentant les racines XML qui satisfont un critère prédéfini. *Suivi* (*follow*) a le même rôle que la projection dans TAX, XAL et l'algèbre relationnelle. Il sélectionne les éléments de chaque collection suivant l'expression de chemin définie. Homologue au concept de sélection dans les autres algèbres, la *sélection* consiste ici à filtrer les ensembles et à conserver seulement ceux qui respectent un prédicat basé sur les opérateurs =, >, <, etc. De même, la *jointure* consiste à rassembler deux collections sur la base d'un prédicat. L'union, l'intersection et le produit cartésien sont aussi définis dans Niagara. Appliqués sur des collections, ils conservent le principe algébrique défini en relationnel. Le *groupement* permet finalement de réunir les éléments correspondants aux chemins indiqués en paramètres. L'ordre de groupement est aussi exprimé comme paramètre d'entrée. Trois nouveaux opérateurs sont également proposés. La *création de sommet* (*vertex*) prend en entrée l'étiquette d'une arête et crée le sommet correspondant en sortie. L'*exposition* (*expose*) permet de rechercher les éléments spécifiques d'une collection, selon le chemin indiqué en paramètre. Enfin, le *renommage* permet de modifier les éléments d'une collection en changeant leurs étiquettes. Les anciennes et les nouvelles étiquettes sont indiquées en paramètres.

2.2.3 Algèbres sur les arbres de données XML pour l'évaluation de XQuery

Les algèbres présentées dans les sections précédentes visent à optimiser les langages de requêtes XML en général. Dans cette section, nous présentons les algèbres sur les arbres XML qui sont conçues plus spécifiquement pour l'évaluation du langage XQuery.

2.2.3.1 Algèbre TLC

Sur la base des notions introduites dans TAX, Paparizos et al. proposent une nouvelle algèbre pour les classes logiques d'arbres nommée TLC (*Tree Logical Class*) [PWLJ04]. Par définition, une classe logique contient l'ensemble des nœuds communs entre les arbres modèles et témoins. Dans TLC, les branches de l'arbre modèle sont annotées par les signes –, +, ? ou *. Cette extension généralise le principe de correspondance (*matching*) entre les arbres modèles et les arbres en entrée. Le signe – indique qu'une et une seule correspondance peut être réalisée. Le signe + correspond à zéro ou plusieurs correspondances. En cas d'utilisation de ?, il est impossible d'aller au-delà d'une seule correspondance, tandis que l'emploi de * autorise un nombre illimité de correspondances.

L'algèbre TLC est conçue pour manipuler l'hétérogénéité au sein des requêtes XQuery. À partir d'exemples réels, les auteurs montrent qu'une requête XQuery peut être vue comme une suite d'opérations logiques. Soit une requête XQuery exemple qui, pour parvenir à son résultat, opère une *sélection*, une *projection*, une *agrégation*, un *groupement*, puis se termine par la *sélection* des éléments visés par l'analyse. TLC permet de traduire cette requête XQuery en une suite d'opérateurs.

L'opérateur de *filtrage* (*filter*) prend en entrée, en plus de la collection d'arbres de données S, un attribut de filtrage p, un mode m et une étiquette LCLf se référant à une classe logique. En sortie, il fournit les sous-arbres de S qui correspondent aux nœuds liés dans la classe logique et qui satisfont p. Le mode m précise comment se fait le choix de l'ensemble de ces nœuds. La *jointure* prend en entrée deux ensembles d'arbres S1 et S2, un arbre modèle annoté apt et un prédicat p. L'annotation utilisée ici est le signe – (une seule correspondance entre l'arbre en entier et l'arbre modèle). L'arbre témoin représente la jointure entre les deux ensembles de nœuds décrits par p. L'opérateur de sélection prend en entrée un ensemble d'arbres S et un modèle apt. La sortie est constituée de l'ensemble des arbres témoins composés des nœuds communs entre S et apt. La projection prend en entrée un ensemble d'arbres et la classe logique représentant les nœuds recherchés. S'ils ne peuvent pas être reliés pour former un arbre, la racine de l'arbre en entrée est employée pour assurer cette liaison. L'*élimination de doublons* (*duplicate-elimination*) prend en entrée une collection d'arbres, un paramètre ci et la liste

des nœuds de la classe logique spécifiée nl. Le paramètre ci élimine les duplications au niveau des noms ou du contenu des nœuds. Un ensemble de fonctions d'*agrégation* est aussi défini dans l'algèbre TLC (*count*, *max*, etc.). En entrée, ces fonctions prennent l'ensemble des arbres modélisant la base, la nature de l'agrégation, les nœuds concernés par l'agrégation et les nœuds cibles à créer. Les nœuds ajoutés à l'issue de l'agrégation doivent avoir les mêmes parents que les nœuds initiaux. Le dernier opérateur de TLC est la *construction* (*construct*), qui prend en entrée une collection d'arbres S et un modèle annoté de construction c. L'arbre c est un modèle annoté normal mais qui permet en plus l'étiquetage, le *renommage* et le *rassemblement* (*jointure*) des arbres de données.

2.2.3.2 Algèbre de requêtes pour les flux de données XML fragmentées

Partant du constat que l'utilisation des données XML distribuées s'accroît continuellement, Bose et al. proposent une algèbre de requêtes pour le langage XQuery qui opère sur des flux de données XML répartis sur le réseau [BFLC03]. Après avoir rassemblé différents fragments de données XML et les avoir représentés selon un modèle disponible sur un serveur, l'algèbre proposée s'applique aux flux de données fragmentées. Les données XML en provenance du client sont modélisées en arbres de données sur le serveur.

Les auteurs se basent sur un de leurs travaux préalables : une algèbre pour les données XML (non fragmentées) stockées sur une seule machine [FLBC02]. Son premier opérateur est l'*extraction*, qui retourne une liste de singletons dont l'unique élément contient l'arbre XML entier. La *sélection*, la *projection*, la *fusion* (*merging*) et la *jointure* suivent la même logique que leurs équivalents en relationnel. Deux opérateurs de parcours de chemins dans les documents XML (*nest* et *unnest*) sont aussi inspirés de l'algèbre relationnelle. La *réduction* (*reduce*) produit le résultat final d'une requête utilisant des fonctions d'agrégation (*count* par exemple). Ces trois derniers opérateurs fonctionnent sur la base de l'opération associative binaire (+) pour additionner les éléments d'un ensemble et les réduire en un seul.

Dans la deuxième algèbre proposée par les auteurs pour les données XML fragmentées, ces opérateurs sont repris et enrichis [BFLC03]. L'*extraction* consiste à extraire les fragments à partir du flux de données et à les identifier via une variable de gamme. L'opérateur de *sélection* filtre les fragments dans le flux selon une condition. La *projection* sélectionne les attributs à partir d'un n-uplet du flux de données en entrée. La *fusion* assure la réunion des n-uplets de deux flux d'entrée en un seul, sans éliminer les doublons. Une *jointure* est possible entre les fragments de deux flux de données différents, selon un prédicat de jointure. Cet opérateur conserve la structure initiale des données. *Unnest* permet de parcourir l'arbre XML selon le chemin et le prédicat indiqués. L'opérateur inverse, *nest*, doit être associé à un groupement, une entête et un opérateur (+). Si l'un des trois n'est pas complètement évalué, l'élément correspondant n'est pas analysé. Finalement, la *réduction* s'applique à l'entête de chaque élément et fusionne les résultats en utilisant l'opérateur (+).

2.2.4 Algèbres XML pour l'ECD

Zhang et Yao présentent une analyse des algèbres XML existantes dans une optique d'extraction des connaissances à partir des données XML [ZY04], par exemple pour découvrir des phénomènes inconnus et utiliser des connaissances acquises à partir de grandes volumes de données (règles d'association, typologies...). En conclusion de cette étude, les auteurs préconisent l'emploi d'une algèbre spécifiquement adaptée. En se basant sur une représentation des données XML en arbres, cette algèbre peut se résumer en cinq opérateurs d'*inclusion* d'arbres, de sous-arbres, de régions (groupes d'arbres), d'enfants ou de chemins.

2.2.5 Discussion

Dans cette section, nous discutons des différentes algèbres XML en reprenant cinq critères proposés par Zhang et Yao dans leur analyse des algèbres XML (cf. Section 2.2.4) : le modèle de données sur lequel se base l'algèbre, l'unité de base d'information manipulée par les opérateurs, les opérateurs disponibles, l'expressivité de l'algèbre et les règles d'optimisation possibles [ZY04].

2.2.5.1 Modèle de données

Le modèle de représentation des documents XML généralement adopté dans la littérature est un arbre de données enraciné, ordonné et étiqueté. Il n'existe pas pour autant de représentation standard.

Dans XAL, chaque document XML est représenté par un graphe enraciné avec une relation partielle d'ordre définie entre ses différentes arêtes. Niagara modélise le document XML en un graphe enraciné et orienté, en indiquant les éléments et les attributs correspondants ainsi que leur ordre par des numéros au niveau des sommets du graphe. TAX et les algèbres qui en sont dérivées traitent les documents XML comme des forêts d'arbres enracinés et étiquetés. Le modèle de Pradhan néglige l'ordre des nœuds du fragment recherché dans l'arbre initial, mais les relie avec le plus proche nœud parent commun et associe à chaque nœud un ou plusieurs mots clés utiles dans la recherche d'information. Une originalité des arbres de Bose et al. est la présence de nœuds vides ou trous (holes), auxquels d'autres sous-arbres enracinés, les remplisseurs (fillers), peuvent être associés dans le but de relier les fragments des documents XML. Finalement, Zhang et Yao exploitent aussi des modèles d'arbres étiquetés dans leur algèbre, sans plus de précision. Finalement, certains auteurs adoptent des représentations ad-hoc. Fernández et al. modélisent les documents XML sous forme de schémas composés de types et de variables globales. Les forêts définies dans ce modèle peuvent être ordonnées ou non. Les forêts non ordonnées sont exploitées pour les jointures, par exemple. Zhang et al. représentent les requêtes XQuery en arbres au sein de l'architecture Rainbow. Selon les besoins d'analyse, certaines données XML conservent leur structure tandis que d'autres sont traduites en relationnel. Enfin, Novak et Zamulin structurent les documents XML dans un modèle de base de données XML basé sur XPath.

2.2.5.2 Unité de base de l'information

Par analogie avec les opérateurs relationnels qui opèrent sur des collections de n-uplets, les opérateurs des algèbres XML agissent sur des collections d'entités. Dans XAL et Niagara, l'entité de base est le sommet (élément ou attribut). Dans ce cas, un opérateur reçoit en entrée et fournit en sortie une collection de sommets. Par contre, les opérateurs de TAX et de ses dérivées, ainsi que l'algèbre pour l'ECD de Zhang et Yao, prennent en entrée et produisent en sortie une collection d'arbres. De plus, un arbre modèle et un arbre témoin accompagnent chaque opération, en entrée et en sortie, respectivement. Selon Pradhan, l'unité d'information de base est le fragment, qui représente une suite de nœuds de l'arbre représentant le document XML. L'unité de base de l'information pour Bose et al. est un arbre enraciné et ordonné, où les trous et les remplisseurs sont pris en compte par certains opérateurs. Les opérateurs de l'algèbre de Fernández et al. manipulent en entrée le contenu d'un document XML (ensemble de types et variables globales) et fournissent en sortie un nouveau document XML. Dans les travaux de Zhang et al., il n'existe pas d'unité d'information de base en raison de la diversité des opérateurs (relationnels, XML ou spécifiques). Finalement, l'entité principale manipulée dans le modèle de Novak et Zamulin est l'expression (séquence de nœuds).

2.2.5.3 Opérateurs

Certaines algèbres ne représentent que la traduction des opérateurs relationnels dans un contexte XML [BFLC03, Pra06]. La plupart des autres étendent les concepts relationnels par de nouveaux opérateurs qui diffèrent d'un travail à un autre. Dans l'ensemble de ces travaux, chaque opérateur est toujours dépendant du modèle de données. Parmi les opérateurs inspirés du relationnel, la *sélection* dans les algèbres XML permet de choisir une entité qualifiée parmi les entités en entrée. La *projection*, en revanche, ne suit pas les principes relationnels et diffère d'une algèbre à l'autre. En effet, en algèbre relationnelle, projeter élimine du résultat les champs non spécifiés. Dans XAL et Niagara, l'opérateur de *projection* est similaire au parcours d'expressions de chemins dans XPath. Dans TAX et ses dérivées, la *projection* consiste à rechercher les entités en entrée de l'opérateur qui correspondent à un modèle défini en sélectionnant les attributs cités dans la liste de projection. La *projection* n'existe pas dans l'algèbre de Pradhan. Par ailleurs, nous soulignons l'importance de certains opérateurs spécifiques à XML comme ceux proposés par Fernández et al. (qui s'expriment par des chemins de type XPath), par Novak et Zamulin (qui sont liés aux expressions FLWOR de XQuery) ou aussi par

Zhang et al. ; ou encore *copier-et-coller* dans TAX, *sommet* dans Niagara et les *opérateurs de construction* de XAL. Les opérateurs pour l'ECD sont également originaux et agissent sur les arbres plutôt que sur les données.

2.2.5.4 Expressivité

Puisque le but d'une algèbre XML est de représenter des requêtes utilisateurs grâce à ses opérateurs, elle est jugée plus performante si elle peut être exprimée dans divers langages de requêtes XML. TAX sort du lot selon ce critère, car elle est supportée par la plupart des langages de requêtes XML tels que XQuery ou XML-QL [DFF+98]. Les opérateurs de Niagara sont exprimés en Quilt [CRF00]. XAL et les algèbres respectives de Pradhan, Bose et al., Zhang et al. ainsi que le binôme Novak et Zamulin supportent les requêtes XQuery. L'algèbre de Fernández et al. constitue un cas particulier, puisqu'elle est elle même conçue comme un langage de requêtes XML. Finalement, la question de l'expressivité ne se pose pas pour l'algèbre pour l'ECD car ses opérateurs sont utilisés exclusivement dans des algorithmes d'ECD.

2.2.5.5 Règles d'optimisation

Comme une seule requête XML peut faire appel à plusieurs opérateurs afin de calculer le résultat attendu, les règles d'optimisation désignent les rôles que prennent ces opérateurs au sein de la requête. Le but d'une optimisation est alors de réduire le nombre de résultats intermédiaires et, par la suite, le nombre d'opérateurs utilisés.

Dans Niagara, les deux opérateurs de *projection* (*follow*), *unnesting follow* et *straight follow*, sont interchangeables. Dans ce cas, le même traitement est effectué, mais la façon de représenter les résultats est différente. Les opérateurs de XAL peuvent être utilisés au sein d'un algorithme d'optimisation de XQuery proposé par les auteurs de cette algèbre. Chaque clause XQuery y est vue comme un ou plusieurs opérateurs XAL. Dans le cadre de TAX, la règle d'optimisation est plus claire avec la traduction des requêtes XQuery en expressions de l'algèbre TLC. Dans l'algèbre de Fernández et al., plusieurs opérateurs peuvent être regroupés et généralisés dans une fonction, qui représente elle-même un opérateur indépendant. Selon les auteurs, ces fonctions peuvent rendre les requêtes plus modulaires et concises. Zhang et al. affirment de leur côté que la combinaison des fonctionnalités de trois types d'opérateurs (relationnels, XML, spécifiques) au sein d'un seul système permet d'atteindre un très bon niveau d'optimisation. Finalement, dans les algorithmes pour l'ECD, plusieurs opérateurs de l'algèbre de Zhang et Yao peuvent être employés à la fois.

2.3 Algèbres XOLAP

Dans cette section, nous nous intéressons aux travaux qui abordent XML et OLAP dans un même contexte. Ces travaux ne présentent pas de véritables algèbres XOLAP complètes, mais forment des tentatives intéressantes de définition d'un environnement XML-OLAP et des notions associées.

Jensen et al. proposent un travail qui se résume en trois points [JMP01]. Ils présentent tout d'abord une architecture d'intégration des données XML et relationnelles. Ils proposent ensuite une modélisation multidimensionnelle des bases de données (XML/relationnelles) OLAP avec UML [OMG07]. Enfin, les auteurs illustrent les considérations nécessaires à prendre en compte pour concevoir des bases de données XML supportant des analyses OLAP. En sortie du système d'intégration proposé, les données sont présentées en relationnel et évaluées avec OQL. Dans la même logique, Niemi et al. présentent une architecture d'intégration de données XML et un modèle de base de données OLAP exploités par un langage dédié, MDX [NNNT02].

Dans ces deux travaux, les opérateurs OLAP spécifiques à XML sont absents, puisque les données OLAP sont exploitées avec des langages de requêtes pour les données relationnelles. Les données XML sont traduites en relationnel avant leur utilisation au sein de ces architectures d'intégration. Pedersen et al. préconisent également d'intégrer des données XML et relationnelles au sein d'un même système [PRP02b, PP03, PPP04]. De plus, ils proposent une approche pour fédérer des sources de données XML avec des cubes OLAP existants. Une fédération se compose d'un cube, d'une collection de documents XML et de liens entre le cube et les documents. Un cube

multidimensionnel est décrit par son nom et ses tables de dimensions et de faits. Chaque dimension est vue comme une hiérarchie de niveaux, où chaque niveau est associé à un groupe de valeurs dimensionnelles. La table de faits est une relation contenant un attribut pour chaque dimension et un attribut pour chaque mesure.

Ce modèle de cube est associé à une algèbre composée de trois opérateurs. L'opérateur le plus fondamental dans une fédération OLAP-XML, la *décoration*, attache une nouvelle dimension à un cube en se basant sur les valeurs des éléments XML liés. La *sélection* sert à filtrer les mesures. L'opérateur de *projection généralisée* (*generalized projection*) a pour rôle l'agrégation des mesures. La projection est vue ici comme une opération de sélection employant une clause de *groupement* (*group by*). Ces opérateurs sont mis en œuvre sur les données OLAP-XML à l'aide une extension de SQLM, nommée SQLXM, qui permet d'associer des requêtes XPath aux requêtes SQLM. SQLM est lui-même une extension de SQL pour le traitement des données multidimensionnelles [PRP02a].

L'ensemble des travaux de Pedersen et al. représente à l'heure actuelle la seule véritable tentative de définition d'une algèbre permettant de réaliser une analyse en ligne OLAP sur des cubes XML. Toutefois, cette algèbre se caractérise par seulement deux opérateurs initiaux exprimés en SQL : la sélection et la projection généralisée, qui correspondent à ce que nous connaissons sous le nom de *slice* et *dice* dans les algèbres OLAP classiques.

3 Algèbre XOLAP proposée

Dans ce chapitre, nous présentons en détail notre algèbre XOLAP : modèle de données, opérateurs et mise en pratique de l'algèbre. Nous présentons au préalable la problématique et l'objectif de notre stage et nous justifions notre choix de l'algèbre TAX pour atteindre nos objectifs. Toutes les figures utilisées dans la présentation des opérateurs de notre algèbre sont disponibles en annexe.

3.1 Problématique et objectif de stage

À ce stade, nous avons, avec en point de mire la définition d'une algèbre XOLAP, dressé un état de l'art que nous espérons le plus exhaustif possible au sujet des algèbres OLAP (cf. Section 2.1) et des algèbres XML (cf. Section 2.2).

Les opérateurs OLAP existants étant définis dans un contexte soit relationnel, soit multidimensionnel, il nous reste désormais à les adapter au modèle de données des documents XML (classiquement un arbre, plus généralement un graphe), ainsi qu'à les enrichir de nouveaux opérateurs spécifiquement adaptés au contexte XML si possible. Les travaux existants qui tendent vers XOLAP (cf. Section 2.3) ne satisfont pas nos objectifs. Ils privilégient en effet la traduction des cubes XML en relationnel et leur interrogation avec des extensions du langage SQL, alors que nous tendons vers une solution native XML exploitant le langage XQuery pour l'interrogation. De plus, ces travaux ne proposent en termes d'algèbre qu'un nombre d'opérateurs très limité.

C'est pourquoi, nous définissons une nouvelle algèbre XOLAP (XML-OLAP) issue de l'extension d'une algèbre XML par les opérateurs OLAP classiques (*construction de cube* (*cube*), *rotation* (*rotate*), *permutation d'une modalité dans une dimension* (*switch*), *forage vers le haut* ou *vers le bas* (*roll up* ou *drill down*), *sélection* et *projection sur un cube* (*slice* et *dice*), *transformation d'une mesure en dimension* ou *d'une dimension en mesure* (*pull* ou *push*)). L'algèbre XML choisie est bien TAX (cf. Section 2.2.2.2), pour des raisons que nous expliquons dans la section suivante. En deuxième lieu, nous essayons de mettre notre contribution en pratique grâce à une interface permettant d'exprimer les opérateurs de notre algèbre en générant les requêtes XQuery correspondantes, appelées aussi des requêtes XQuery décisionnelles, qui seront exécutées au sein d'un SGBD natif XML.

3.2 Justification du choix de TAX

Face au grand nombre d'algèbres XML proposées dans la littérature, notre choix s'est porté sur TAX. Notre motivation principale provient de la richesse de cette algèbre qui comprend, sous ses formes logique et physique, plus qu'une vingtaine d'opérateurs. Cela nous offre une grande liberté pour les combiner et exprimer les différents opérateurs OLAP que nous souhaitons mettre en œuvre. De plus, l'expressivité de TAX est largement reconnue, puisque cette algèbre peut être exprimée avec la majorité des langages de requêtes XML, et notamment XQuery, qui est le langage que nous ciblons pour des analyses OLAP. Enfin, TAX et son algèbre dérivée TLC fournissent un cadre d'optimisation de requêtes que nous pourrons exploiter, car la performance est un de nos principaux soucis pour des applications décisionnelles intégralement basées sur XML et XQuery.

3.3 Algèbre XOLAP

3.3.1 Modèle de données

Avant d'exposer notre algèbre XOLAP, et comme pour les algèbres étudiées dans l'état de l'art, nous commençons par présenter le modèle de données sous-jacent qui est, logiquement, inspiré de celui de TAX.

Nous nommons un fichier XML modélisant des données multidimensionnelles, connu aussi sous le nom d'un entrepôt de données XML, un fichier XML multidimensionnel. Normalement, la modélisation d'une table de faits (ou d'un fait) avec XML ne pose pas de problème. Nous rappelons qu'un fait est composé d'une ou plusieurs mesures (valeurs numériques) et des dimensions qui les décrivent. Par conséquence, les occurrences d'un même élément du fichier XML multidimensionnel

modélisent un fait. Les sous-éléments de chaque fait XML représentent alors ses dimensions et ses mesures. Un exemple d'un fait XML *vente* de l'entrepôt *ventes* est disponible dans la figure 1.

Les fichiers XML multidimensionnelles sont modélisables en cubes OLAP, que nous appelons cubes XOLAP, et en arbres TAX puisque ces derniers modélisent tout fichier XML. Les cubes XOLAP, comme les cubes OLAP, sont vus sous la forme de *n* axes, où chaque axe modélise une dimension dans un repère n-dimensionnel. Chaque document XML multidimensionnel peut être traduit à la fois en arbre TAX et en cube XOLAP. Nous pouvons aussi représenter des données sous ces deux formes en fichiers XML multidimensionnels. Une transformation des arbres TAX représentant des données XML multidimensionnelles en cubes XOLAP est donc possible, et vice-versa. Comme nous choisissons d'étendre TAX par des opérateurs OLAP, nous adoptons le modèle de données utilisé par Jagadish et al. [JLST01], les arbres TAX, pour les données XML multidimensionnelles. Dans un arbre TAX modélisant un fichier XML multidimensionnel, que nous appelons aussi un arbre TAX multidimensionnel, chaque sous-arbre représente un fait (sous-élément).

Nous utilisons l'exemple suivant dans le reste de notre travail. Soit un fichier modélisant un entrepôt de données XML *ventes*. Les mesures représentent les prix des produits vendus des différentes villes et à différentes années. Nous disposons alors de trois dimensions, *ville*, *produit* et *année*, et une mesure *prix*, modélisés dans un élément fait intitulé *vente*. Chacune des dimensions est caractérisée par trois modalités : *Paris*, *Lyon* et *Nice* comme *villes* ; *Ecran*, *Souris* et *Clavier* comme *produits* ; et *2004*, *2005* et *2006* comme *années*. En liaison avec ce que nous avons dit précédemment dans cette section, chaque fait XML est modélisable sous forme d'arbre TAX et de cellule XOLAP (figure 2). Supposons maintenant que *ventes* représente vingt-sept faits *vente*. Le fichier XML multidimensionnel est alors constitué d'une racine principale *ventes*, et de vingt-sept occurrences de *vente*. Chaque *vente* précise les dimensions et les mesures correspondantes. La structure du cube XOLAP équivalent à ce fichier XML est illustrée dans la figure 3. En outre, l'arbre TAX équivalent à ce cube et au fichier XML est aussi constitué de vingt-sept sous-arbres *vente* réunis sous une seule racine *ventes*. En cas d'entrepôt creux (une ou plusieurs mesures absentes), une cellule vide est représentée de la même manière, dans le fichier XML comme sous forme d'arbre TAX, tout en en donnant une valeur nulle à la mesure, toujours présente avec les dimensions correspondantes.

Pour améliorer le modèle de données utilisé, nous définissons un sens de lecture standard des données XML multidimensionnelles selon les trois modes de représentation possibles : fichiers XML multidimensionnels, cubes XOLAP et arbres TAX multidimensionnels. Le sens de lecture pour les deux premières formes est de haut en bas (la mesure est lue en dernier). Pour les arbres, le sens de lecture est de gauche à droite. Les axes représentant les dimensions suivent un sens de lecture adaptée, de haut en bas puis de gauche à droite.

Dans ce qui suit, nous présentons les différents opérateurs de notre algèbre XOLAP. Nous mettons en œuvre chaque opérateur sur les données multidimensionnelles de la base *ventes*, citée dans la section précédente et modélisée en arbres TAX. Pour chaque opérateur, nous donnons sa définition originale dans l'algèbre OLAP classique, et nous l'exprimons dans un contexte XOLAP via un ou plusieurs opérateurs TAX. Pour cela, nous nous sommes basés essentiellement sur un ensemble de supports de cours sur l'OLAP [Sho97, Adi05, Van05, Ben07, Gar07] en plus de l'algèbre TAX.

3.3.2 Opérateurs

3.3.2.1 Opérateurs liés à la structure

- **Rotation (rotate)**

La *rotation* (*rotate*) ou *pivotement* (*pivot*) [Gar07] permute les dimensions d'un cube en lui faisant subir une rotation autour d'un de ses axes passant par le centre de deux faces opposées, de manière à présenter un ensemble de faces différents dans le cube. Une *rotation* est une opération visuelle qui consiste déplacer deux dimensions ou plus sans modifier les valeurs des mesures correspondantes. D'un point de vue technique, une *rotation* consiste en une sélection de faces et non de membres. Le résultat d'une *rotation* de notre cube XOLAP exemple autour de l'axe produit (figure 3) est disponible dans la figure 4. Une représentation axiale de ce même exemple est illustrée dans la figure 5.

Un exemple de *rotation* d'un fait du fichier XML multidimensionnel, suivant l'axe produit, est disponible dans la figure 6. Il s'agit en fait d'interchanger les positions des dimensions autres que la dimension, ou l'axe, de rotation. Pour exprimer une *rotation* avec TAX, nous avons recours à la *sélection*. Une *sélection* TAX prend à son entrée l'arbre modélisant l'entrepôt de données XML, un arbre modèle et une liste de sélectionss. Suivant le sens de lecture défini précédemment (cf. Section 3.3.1), l'ordre des dimensions dans l'arbre modèle doit correspondre à l'ordre des axes dans le résultat. La liste de sélectionss contient obligatoirement le nœud *vente* afin de garantir la préservation de tous ses descendants pour l'ensemble des occurrences en sortie de la sélection. Dans une étape suivante, il nous reste à réunir tous les arbres sélectionnés sous une seule racine *ventes* via l'opérateur TAX *production*. Le résultat de la rotation d'un fait XML sous ses trois formes (arbre, fichier XML et cellule XOLAP) est disponible dans la figure 6.

- **Changement de position des modalités dans une dimension (switch)**

L'opération de *changement de position des modalités dans une dimension* (*switch*) consiste à interchanger les positions de deux ou plusieurs attributs au sein d'une même dimension. De même que la *rotation*, cette opération se caractérise par un effet visuel en résultat, puisque les valeurs des mesures ne changent pas avec le changement de position des faits correspondants. Un exemple de *changement de position des modalités dans une dimension* dans notre cube XOLAP initial est disponible dans la figure 7. Nous y permutons les attributs *Nice* et *Lyon* de la dimension *ville*. En respectant le sens de lecture des données XOLAP que nous avons défini, et quelle que soit la forme de leur présentation multidimensionnelle, nous présentons trois fragments de données XML inspirés du cube précédent et les arbres correspondants dans la figure 8.

Pour réaliser un *changement de position des modalités dans une dimension* en utilisant TAX, il suffit de réordonner les sous-arbres de l'arbre TAX multidimensionnel en sortie. L'opérateur de *réordonnancement* TAX paraît le mieux adapté pour atteindre cet objectif. Le *réordonnancement* prend en entrée la collection d'arbres modélisant les données, un modèle d'arbre, une fonction de valeur d'arbre et une liste de réordonnancement. Le modèle d'arbre précise les nœuds à réordonner. L'ordre que prennent ces nœuds est indiqué dans la liste de réordonnancement. La fonction de valeur indique la valeur que prend chaque nœud participant dans l'ordre général. Ces trois derniers éléments sont illustrés dans la figure 9. Comme notre but est d'inter-changer les villes *Nice* et *Lyon*, *Paris* paraît écarté de l'analyse. Pourtant, l'attribut correspondant à cette ville doit être indiqué dans l'arbre modèle pour que *Paris* apparaisse dans le résultat, même s'il conserve son emplacement. En d'autres termes, réordonner les attributs d'une dimension consiste à indiquer à la fois l'ordre des attributs concernés et non concernés puisque ces derniers participent à l'ordre final.

Nous passons maintenant à deux autres opérateurs structuraux, la *transformation d'une dimension en mesure* (*push*) et la *transformation d'une mesure en dimension* (*pull*). Dans les algèbres OLAP discutées dans l'état de l'art (cf. Section 2.1), nous ne retrouvons pas les résultats de ces deux opérations sous la forme de cubes OLAP, mais modélisés dans des tables multidimensionnelles. Pourtant, Agrawal et al. exposent ces deux opérateurs sur leur modèle de données de repères à trois dimensions modélisant les cubes de données [AGS96, AGS97]. Nous avons donc recours à ce travail, puisque nous soutenons l'idée de représenter les cubes en repères n-dimensionnels dans notre algèbre.

- **Transformation d'une dimension en mesure (push)**

La *transformation d'une dimension en mesure* (*push*) consiste à combiner les membres d'une dimension aux mesures du cube, c'est-à-dire à faire passer ses membres comme contenu de cellules [Ben07]. Nous voyons dans la figure 10 comment Agrawal et al. expriment la *transformation d'une dimension en mesure* en associant à la mesure *Sales* la dimension cible par l'opération *Product*. La fonction de la *transformation d'une dimension en mesure* affiche les faits d'une façon adaptée à la définition de l'opération elle-même, sans modifier les valeurs des mesures.

Dans un contexte TAX, il suffit alors de restructurer les sous-arbres faits, de manière à adapter la sortie au principe de la *transformation d'une dimension en mesure*. Ici, et en nous inspirant du

travail d'Agrawal et al., nous associons pour chaque fait, la dimension ciblée par l'opération à la mesure.

Soit l'exemple de la figure 11 illustrant une *transformation de la dimension* ville *en mesure* prix. Là dedans, nous remarquons que le nœud représentant la mesure est développé de façon à contenir à la fois *prix* et *ville* comme éléments fils au même temps.

Pour réaliser ce but, nous avons recours à l'opération TAX *copier-et-coller* qui prend en entrée un modèle d'arbre, une liste de copie et une liste de spécifications. Le modèle d'arbre permettant de transformer la dimension ville en mesure est disponible dans la figure 12. La liste de spécifications indique que les arbres insérés sont les descendants directs du nœud prix (*$4* dans le modèle). La liste de copie détermine les différentes valeurs des nœuds apparaissant en sortie. En résultat, nous obtenons le résultat de la figure 13.

- **Transformation d'une mesure en dimension (pull)**

Pour exprimer l'opération réciproque de la *transformation d'une dimension en mesure*, Agrawal et al. expriment la *transformation d'une mesure en dimension* en représentant, suivant leur modèle, l'ensemble de mesures sur un axe pour la transformer en mesure [AGS96, AGS97]. En réalité, Agrawal et al. partent d'une représentation du résultat d'une *transformation d'une dimension en mesure* pour arriver à l'état initial en permutant les emplacement de la mesure et la dimension y associée, comme le montre la figure 14. Nous ne suivons pas leur logique, bien qu'elle soit juste dans le sens où elle est adaptée à leur modèle, car nous devons suivre les exigences du nôtre. Nous exprimons alors la *transformation d'une mesure en dimension* avec TAX comme suit.

Si nous adoptons la manière d'exprimer la *transformation d'une dimension en mesure* et que nous associions la mesure à une dimension comme nous avons associé une dimension à une mesure, nous commettrions une erreur. En fait, selon le modèle de nitre algèbre, une mesure est toujours représentée, si elle existe, comme la dernière branche du sous-arbre modélisant le fait en allant de droite à gauche. Représenter la mesure avec toutes les dimensions permet selon notre modèle de la considérer comme une dimension. Il suffit alors de réaliser une *projection* sur tous les sous-arbres de l'arbre de base. L'arbre modèle et la liste de projection, en entrée de l'opération, indiquent la nouvelle position que prenne la mesure, parmi les dimensions, qui permet sa transformation en dimension en sortie. Un exemple de *transformation d'une mesure en dimension* avec TAX est disponible dans la figure 15.

Nous n'avons pas procédé de la même façon pour la *transformation d'une dimension en mesure* puisqu'une mesure doit exprimer une valeur numérique. Par contre, il n'est pas interdit qu'une dimension exprime aussi une valeur numérique, ce qui est le cas suite à une *transformation d'une mesure en dimension*. En d'autres termes, une dimension peut être représentée à la fin de l'arbre si cette dernière ne dispose pas de mesure. Mais représenter une mesure avec les dimensions nous pose dans un cas spécial, celui de la *transformation d'une mesure en dimension*.

3.3.2.2 Opérateurs ensemblistes

Dans ce qui suit, nous exprimons les deux opérateurs ensemblistes de l'OLAP *slice* et *dice* dans un contexte XOLAP. Connus généralement sous les noms de *sélection* et *projection* sur un cube, respectivement, nous adoptons d'autres dénominations plus précises, inspirées essentiellement de leurs principes, *découpage de cube en tranches* pour *slice* et *détachement d'un dé du cube* pour *dice*. Notre utilisation de ces nouveaux termes est justifiée dans la présentation de ces deux opérations.

- **Découpage de cube en tranches (slice)**

Le *découpage de cube en tranches* (*slice*) dissocie une ou plusieurs tranches d'un cube de données multidimensionnel selon un ou plusieurs attributs d'une seule dimension, et ce pour tous les attributs dimensionnels correspondants. Un cube peut d'ailleurs être vu comme un ensemble de tranches superposées les unes sur les autres (horizontalement), ou les unes à coté des autres encore (verticalement). Donc, *le découpage de cube en tranches* peut s'effectuer dans l'un de ces deux sens,

mais pas dans les deux simultanément. Nous morcellerions alors le cube en cellules (ou groupes de cellules), ce qui n'est pas le but ici.

Techniquement, un *découpage de cube en tranches* est assimilé à une *projection* sur un cube de données dans le sens de la réduction dimensionnelle (et non dans celui de l'algèbre relationnelle) bien que les deux soient conceptuellement homologues [Gar07, Wik07]. Cette *projection* est plus claire en cas d'extraction de toutes les tranches d'un cube, de manière à ce que l'union des tranches résultantes constitue le cube de départ. Cette opération nous rappelle l'opération SQL *select * from table*, qui détermine une *projection* via une *sélection*. En réalité, une *projection* peut être vue comme une *sélection* dans SQL comme dans l'OLAP. Pourtant, le raisonnement inverse (de *sélection* à *projection*) est faux. D'ailleurs, la *sélection* est le cas général de toute opération de *projection*, dans l'algèbre relationnelle comme dans l'algèbre multidimensionnelle. C'est principalement pour toutes ces raisons que nous n'avons pas adopté *sélection*, ni même *projection*, pour nommer cette opération.

Dans l'exemple de la figure 16, nous voyons qu'un *découpage de cube en tranches* écarte du résultat l'ensemble des tranches non visées par l'analyse. Autrement dit, un tel opérateur élague du résultat l'ensemble de faits en commun entre les attributs dimensionnels de l'axe à lequel appartient l'attribut dimensionnel selon lequel se réalise la coupe, sauf ce dernier, et tous les autres attributs des autres axes.

Nous étendons TAX par l'opérateur de *découpage en tranches de cubes en tranches* comme suit. L'objectif est l'extraction, depuis l'arbre initial, des sous-arbres correspondants à l'ensemble de tranches visées par l'opération. L'arbre représentant le résultat de *découpage* de l'arbre de la figure 16 *en tranches* est disponible dans la figure 17. Un *découpage de cube en tranches* avec TAX est assuré en sélectionnant tous les faits correspondants aux tranches à conserver. Ici, la *sélection* TAX prend en entrée l'arbre modélisant la base, un arbre modèle et une liste de paramètres. Puisque la coupe se réalise suivant un ou plusieurs attributs d'une seule dimension, il suffit de les indiquer dans l'arbre modèle. Par exemple, pour atteindre le résultat de la figure 17, l'attribut élément selon lequel se réalise la coupe, *Ecran*, est associé à la définition de l'arbre modèle, comme le montre la figure 18. La liste de sélections doit contenir obligatoirement le nœud *$1 ($1.tag=vente*) pour garantir la conservation de tous ses éléments fils dans le résultat. À l'issue de cette *sélection*, nous obtenons un ensemble d'arbres *vente* séparés que nous réunissons tous sous une seule racine *ventes* via une *production* TAX.

- **Détachement d'un dé du cube (dice)**

Le *détachement d'un dé du cube* retire sous-cube du cube de données multidimensionnel selon certains prédicats définis sur ses dimensions. Certaines références considèrent que le *détachement d'un dé du cube* est une *projection* sur une ou plusieurs dimensions du cube [Ben07]. D'autres la considèrent comme une *sélection* dans le sens de la réduction dimensionnelle, pas dans le sens relationnel [Gar07, Wik07].

Pour notre part, nous partons du principe général de l'opération consistant à sélectionner les mesures d'un cube pour un certain nombre d'attributs dimensionnels, qui doit être le même pour tous les axes du cube. De plus, les cellules du dé résultant peuvent être attachées ou non ensemble dans le cube original [Adi05]. Un exemple de *détachement d'un dé du cube* est disponible dans la figure 19, dans lequel nous conservons de notre cube XOLAP exemple que les ventes des *souris* et des *claviers* à *Lyon* et *Nice* en *2004* et *2005*.

Pour exprimer un *détachement d'un dé du cube* avec TAX, nous sélectionnons les sous-arbres correspondants aux faits représentants le dé à extraire. Nous exprimons cette opération par une *sélection* TAX de tout sous-arbre (fait) représentant une cellule. Cette *sélection* prend en entrée l'arbre base initial, l'arbre modèle de la figure 20 et une liste de sélections contenant le nœud *$1* (*vente*) pour conserver ses descendants. Les clauses *OR* utilisées dans ce modèle n'existent dans un aucun modèle de l'algèbre TAX. Pourtant, Jagadish et al. emploient une clause *BEFORE* dans un modèle de leur travail [JLST01]. Désormais, ils ont pu utiliser une clause *AFTER*, par exemple, à la place de *BEFORE* en inversant juste l'ordre de paramètres correspondants. Si *OR* n'est pas employée au modèle de la figure 20, nous avons recours à plusieurs modèles à la fois. À l'issue de la *sélection*, nous obtenons les huit arbres équivalents aux huit cellules du dé de la figure 19. Il reste alors à les réunir sous une même racine *ventes* avec une *production* TAX.

3.3.2.3 Opérateurs liés à la granularité

Les opérateurs liés à la granularité considèrent, dans leur logique, les hiérarchies des dimensions en représentant les données multidimensionnelles. La granularité d'une dimension désigne le nombre de niveaux hiérarchiques qu'elle puisse représentée cette dimension [Van05].

- **Forage vers le haut (roll-up)**

Le *forage vers le haut* (*roll-up*) représente les données du cube à un niveau de granularité supérieur selon une dimension, conformément à sa hiérarchie. Le but est de passer d'un niveau de granularité à un autre niveau plus général, par exemple, passer d'une représentation multidimensionnelle des données suivant les villes à une autre suivant les pays. Ce passage nécessite l'association d'une fonction d'*agrégation* (*somme, moyenne*, etc.) à l'opération de *groupement*. Le but de cette association est d'indiquer comment sont calculées les valeurs du niveau supérieur à partir de celles du niveau inférieur.

Techniquement, un *forage vers le haut* consiste à aplatir (ou plier) les différentes tranches du cube en une seule tout en suivant la direction de l'axe représentant la dimension concernée par cette opération. Chaque cellule du résultat final contient le résultat de l'agrégation des mesures correspondantes. Un exemple du *forage vers le haut* sur notre cube XOLAP est présenté dans la figure 21, où nous calculons le total des ventes sur toutes les années pour chaque *groupement* (produit, ville).

Dans un contexte relationnel, une fonction d'*agrégation* est souvent associée à un *groupement*. L'emploi du *groupement* précise les attributs visés par l'*agrégation*. Dans TAX, Jagadish et al. séparent l'*agrégation* du *groupement* dans TAX. D'autres approches optent pour cette idée et proposent une algèbre OLAP où le *forage vers le haut* est exprimé en associant une *jointure* à une *production* [CAJK01].

Pour notre part, nous choisissons de garder le principe de l'association *groupement-agrégation* pour réaliser un *forage vers le haut* avec TAX. *Groupement* et *agrégation* consistent en deux opérations TAX complètement distinctes, mais nous pouvons néanmoins les combiner dans un même contexte, tout en respectant les principes de chacun. L'arbre correspondant au résultat que nous voulons atteindre, et qui modélise le cube résultant du forage de la figure 21, est disponible dans la figure 22. Il comporte neuf sous-arbres qui représentent chacun un fait *vente* décrit par le lieu de vente et le produit vendu. La dimension *Temps* combine toutes les valeurs que peut prendre une année. La fonction d'agrégation choisie dans notre exemple est la *somme* (*sum*). Chaque mesure représente alors la *somme* des ventes durant les trois années, pour tout *groupement* (ville, produit) possible, en suivant le même axe que dans le cube original. Dans ce qui suit, nous démontrons comment une suite d'opérateurs TAX permet d'aboutir, par étapes, à un *forage vers le haut*.

1. Sélection : Il faut en premier lieu extraire tous les sous-arbres avec une *sélection* TAX avec un modèle et une liste de sélections conservant les structures et les contenus initiaux de tous les arbres de l'entrepôt. Le but de cette *sélection* est de détacher tous les arbres afin de les préparer au processus suivant.

2. Groupement : Nous groupons ensuite les sous-arbres détachés d'une façon adaptée à la définition du *forage vers le haut*. Le *groupement* est réalisé suivant toutes les combinaisons possibles des attributs des dimensions autres que celle du forage. Le *groupement* prend en entrée les arbres issus de la sélection, un modèle de groupement, une fonction de groupement et une fonction d'ordre de valeur d'arbre. Le modèle de groupement précise que cette opération est assurée par les attributs de la dimension autres que celle du forage, année dans notre cas. Le modèle correspondant est disponible dans la figure 23. La fonction de groupement précise l'ordre des éléments dans les sous-arbres du résultat. La fonction d'ordre indique l'ordre que prennent ces sous-arbres. Dans notre exemple, l'ordre des sous-arbres ainsi que de leurs nœuds est bien l'ordre départ. La figure 24 montre un exemple de *groupement* des ventes par ville et produit. Au total, nous obtenons neufs arbres selon toutes les combinaisons des valeurs possibles (ville, produit). Avant de passer à l'étape suivante, il faut faire une *sélection* de tous les sous-arbres selon un modèle qui prend comme racine *tax_group_subroot*.

3. Jointure : Pour chaque sous-arbre *tax_group_subroot*, exprimant un *groupement*, nous réalisons une *jointure* TAX des trois sous-arbres correspondants en un seul arbre selon le modèle de la figure 25. Le résultat de *jointure* des arbres de la figure 24 est présenté dans la figure 26. Il suffit ensuite de sélectionner depuis chaque résultat de jointure l'arbre *vente* en utilisant le modèle d'arbre et la liste de sélections correspondants. La jointure peut être réalisée aussi selon les années (l'axe qui n'a pas participé au *groupement*). Pourtant, nous n'avons pas choisi cette solution et nous avons procédé comme décrit déjà.

4. Agrégation : L'opération d'*agrégation* prend en entrée une fonction d'agrégation, un modèle d'arbre et un ensemble de spécifications de mise à jour. La fonction d'agrégation utilisée dans notre exemple est *somme* (*sum*). Le modèle d'arbre doit conserver tous les nœuds de l'arbre d'entrée, avec en plus un nouveau nœud *$i* supportant l'agrégation calculée. La spécification de mise à jour précise comment se réalise l'agrégation. Ici, un nœud *$i* prend comme valeur la *somme* du reste de nœuds mesures. Un résultat d'*agrégation* sur le sous-arbre *vente* de la figure 26 est présentée dans la figure 27. Dans les arbres résultants, la présence des nœuds mesures autres que *$i* est inutile. Pour les élaguer, nous avons recours à une *projection*. Le modèle et la liste de projection correspondants ne spécifient pas ces nœuds comme résultat souhaité.

5. Insertion de nœuds : Nous employons l'opération d'*insertion de nœuds* pour insérer dans chaque arbre un nœud représentant la dimension *Temps*, qui agrège les trois années. À l'issue de cette insertion, il reste à associer les neufs arbres *vente* sous une seule racine *ventes* via une *production* TAX.

- **Forage vers le bas (drill-down)**

Le *forage vers le bas* est l'opération réciproque du *forage vers le haut*. Le *forage vers le bas* permet d'obtenir des détails sur la signification d'un résultat en affinant une dimension à un niveau d'agrégation inférieur (par exemple, passer d'une représentation multidimensionnelle des données suivant les pays à une autre représentation suivant les villes). D'un point de vue technique, si le *forage vers le haut* permet de plier un cube, le *forage vers le bas* le déplie en détaillant les données suivant une (ou plusieurs) dimension.

Un exemple de *forage vers le bas* est illustré dans la figure 28, où les faits sont détaillés selon la dimension ville. Le but est de détailler les ventes par arrondissement, pour tous les produits et pour toutes les années. La fonction d'agréation choisie ici est la *somme*. Généralement, nous pouvons choisir un autre type d'agrégation, *moyenne* par exemple. En sommant les valeurs des ventes d'un produit pour les arrondissements d'une même ville au cours d'une année donnée, nous devons obtenir la valeur de la vente dans le cube initial pour ces mêmes dimensions. Chaque fait doit passer dans le résultat de sa modélisation par un arbre à un certain nombre d'arbre selon les exigences du forage. Pour chaque fait, le nombre d'arbres en sortie du *forage vers le bas* est multiplié en le nombre de modalités que prend la dimension élément de forage. L'exemple de la figure 29 illustre cela.

Le *forage vers le bas* nécessite des données supplémentaires pour sa mise en œuvre. En d'autres termes, les données représentées en sortie sont supposées connues par le manipulateur. Dans l'exemple présenté précédemment, les arrondissements de chaque ville ainsi que les mesures correspondantes sont supposés connues. Bien qu'il soit l'opération réciproque du *forage vers le haut*, nous n'exprimons pas le *forage vers le bas* en utilisant le cheminement réciproque des opérateurs utilisés. Nous ne réalisons pas en effet une *agrégation* suivie d'un *groupement*. Grâce à la richesse de TAX, nous pouvons utiliser un minimum d'opérateurs pour assurer un *forage vers le bas* comme suit.

1. Suppression de nœuds : La première opération à effectuer est de supprimer les arbres représentant les faits (sous-arbres) à détailler, afin de les substituer par les faits correspondants. Une *suppression de nœuds* prend en entrée, dans TAX, l'arbre modélisant l'entrepôt, un modèle d'arbre et une spécification de suppression indiquant les nœuds à élaguer de l'arbre. Dans notre exemple, l'entrepôt représente la représentation en arbres TAX du cube XOLAP de la figure 29. Le modèle d'arbre indique le format d'arbres à supprimer tout en indiquant, obligatoirement, l'attribut élément de la suppression (figure 30). Le signe * dans *$1** indique que le nœud *vente* doit être supprimé avec l'ensemble de ces fils. Dans notre cas, le résultat final est un arbre vide ne contenant aucun sous-arbre,

puisque tous les faits sont à substituer. Le nœud racine *ventes* reste, par conséquent, sans descendant pour le moment.

2. Copier-et-coller : Les nœuds supprimés à l'étape précédente doivent être substitués par d'autres. Nous pourrions utiliser alors l'opération TAX *insertion de nœuds*. Cependant, selon Jagadish et al., il est recommandé d'employer l'opérateur *copier-et-coller* en cas d'insertion de nœuds accompagnés de leurs descendants [JLST01]. Dans notre exemple, nous copions et collons à la place de chaque arbre supprimé deux arbres contenant chacun les informations sur les ventes pour les arrondissements correspondants. Par exemple, à la place de l'arbre des ventes des souris à Paris en 2004, il s'agit de copier et coller les deux arbres de la figure 29 modélisant les ventes des souris dans les 1er et 8ème arrondissements.

Copier-et-coller prend en entrée un modèle d'arbre, une liste de copie et une liste de spécifications. La liste de copie indique les différentes valeurs que prendront les nœuds apparaissant dans le modèle en sortie. Ainsi, la liste de spécifications indique que les arbres insérés sont les descendants directs du nœud ventes issu de l'opération précédente de suppression de nœuds. Tous ces éléments sont disponibles dans la figure 31. À la fin, nous obtenons seize arbres qui possèdent chacun une cellule équivalente dans le cube de la figure 28.

- **Construction de cube (cube)**

Dans un premier travail, Gray et al. définissent l'opérateur *cube de données* (*data cube*), ou *construction de cube* (*cube*), comme un opérateur relationnel généralisant le *groupement* (*group-by*), le *produit de tables* (*cross-tab*) et le *forage vers le haut* (*roll-up*) [GBLP96]. Un an plus tard, ils assimilent *cube de données* (*data cube*) à un opérateur *d'agrégation relationnel* généralisant le *groupement* (*group-by*), le *produit des tables* (*cross-tab*) et les *sous-totaux* (*sub-totals*) [GCB+97]. Les *sous-totaux* désignent, en général, les *agrégations* utilisées dans le *forage vers le haut* (c'est à dire les *totaux*), ou le *forage vers le bas* (ici, nous utilisons les *sous-totaux*). *Cube* groupe les résultats de différentes *agrégations* réalisées sur un cube, en tenant compte de tous les niveaux de granularité possibles des données de ce cube. L'utilisation du *produit des tables* (*jointure*) permet d'avoir un cube plus grand groupant les résultats et les agrégations appliquées.

Nous exprimons *cube* avec TAX grâce à l'enchainement des opérateurs suivants. Ces différentes étapes sont illustrées la figure 32. Le cube de données initial choisi dans cette partie est celui de la figure 32-1.

1- Forage vers le haut : En premier lieu, nous réalisons *N forages vers le haut* TAX sur le cube de données exemple (figure 32-2) selon le même principe décrit précédemment (cf. Section 3.3.1). *N* désigne le nombre d'axes représentants le cube. Dans notre cas, trois *forages vers le haut* sont effectués, suivant les axes (dimensions) produit, année et ville. L'*agrégation* choisie dans les trois forages est *somme*. Le résultat final des trois *forages vers le haut* est disponible dans la figure 32-2.

2- Groupements, agrégations et jointures : Pour chaque résultat du *forage vers le haut*, nous réalisons un *groupement* pour tous les attributs des dimensions participant à ce forage. Par exemple, pour le résultat du *forage vers le haut* suivant l'axe produit, nous réalisons les *groupements* suivants : deux *groupements* par ville, un pour l'année 2003 et l'autre pour 2004, et deux autres groupements par année et par ville (Paris et puis Lyon). Dans les différents *groupements*, nous conservons la même fonction d'agrégation (*somme* ici). Les résultats des *groupements* sont explicités dans la figure 32-3. Par exemple, l'arbre *v21* représente le résultat du *groupement* des produits *Ecran* par ville pour les années 2003 et 2004. Les autres arbres (de *v22* jusqu'à *v32*) suivent aussi cette logique. La mesure du fait *v21*, 400, est la somme de 210 et 190 (les valeurs des mesures des arbres *v9* et *v10*, respectivement), ce qui démontre la validité de notre démarche jusqu'à ici. Dans TAX, pour chaque *groupement*, l'arbre modèle spécifie les deux dimensions éléments du *groupement*. Nous sommons ensuite, pour chaque arbre résultant du *groupement*, les deux mesures en une seule, en ayant recours à l'opérateur d'*agrégation* TAX.

Nous remarquons dans la figure 32-3 que quelques résultats de *groupements* se répètent. Par exemple, le couple d'arbres (*v21, v22*) est similaire à (*v25, v26*). Nous remarquons que ces deux combinaisons de sous-arbres représentent une dimension commune, *produit*. L'emplacement des

cellules correspondantes dans le cube final est alors effectué suivant cet axe. In es est de même pour les combinaisons ((*v23, v24*) ; (*v31, v32*)) -l'axe en commun ici est *ville-*, et ((*v27, v28*) ; (*v29, v30*)) - l'axe en commun ici est *annee-*. Sur le schéma du cube, il suffit de représenter une des deux combinaisons pour chaque axe, puisque les valeurs de mesures se répètent chez la combinaison équivalente.

En faisant la correspondance des cellules avec les arbres de faits TAX, il suffit de joindre, via une *jointure* TAX, les quatre arbres des deux ensembles selon leur axe commun entre eux. Par exemple, nous joignons les arbres *v21, v22, v25* et *v26* selon la dimension *produit*. De même pour ((*v23, v24*) ; (*v31, v32*)) -*jointure* selon les *villes*- et ((*v27, v28*) ; (*v29, v30*)) -*jointure* selon les *années*- . La complétion du cube de la figure 32-3 est illustrée dans la figure 32-4.

En arrivant au résultat de la figure 32-4, nous remarquons qu'une seule cellule manque pour constituer le cube final. En effet, cette cellule correspond au point de contact entre les trois résultats de *jointure*. La mesure contenue dans cette cellule représente le résultat de l'*agrégation somme*, des ensembles de mesures du résultat de *jointure*. En fait, le résultat de cette *agrégation* est le même quel que soit l'axe choisi pour cette opération. Il suffit alors de sommer deux cellules en suivant un même axe pour obtenir le bon résultat. Dans TAX, obtenir l'arbre équivalent à cette cellule se fait en joignant, une deuxième fois, chacun des deux arbres résultants de la *jointure* selon chaque axe, et en appliquant une *agrégation* de type somme sur chaque arbre. Et puisque nous obtenons trois résultats identiques dans les trois cas, il ne reste qu'à grouper les trois arbres en un seul. Comme le *groupement* élimine les doublons du résultat, une seule mesure est conservé à la fois dans le résultat final alors. Nous justifions le choix du *groupement* par le fait que l'arbre correspondant à cette dernière cellule est exprimé selon les trois axes au même temps tout en représentant le plus haut niveau d'agrégation sur le cube. Sur le cube OLAP de la figure 32-4, la cellule courante est celle qui représente la mesure 685.

En achevant toutes les étapes de l'expression de l'opérateur de *cube de données*, nous obtenons un arbre TAX, contenant les 32 sous-arbres issus de toute cette démarche. Chaque sous-arbre correspond à une cellule de cube de données de la figure 32-4.

3.4 Discussion

Dans cette section, nous discutons l'algèbre XOLAP présentée selon deux critères. Comme notre algèbre est une extension de TAX, elle en a hérité le critère de règles d'optimisation. De plus, chaque opérateur XOLAP respecte dans sa définition celle de son équivalent en OLAP classique.

3.4.1 Règles d'optimisation dans notre algèbre

TAX a montré sa fiabilité surtout en termes d'adaptabilité à toutes formes de données XML, comme les données multidimensionnelles dans notre cas. Le critère qui a contribué à la création de nos opérateurs XOLAP est des règles d'optimisation offertes par TAX (cf. Section 2.2.5.5). Chaque opérateur XOLAP est constitué d'une combinaison de plusieurs (au moins deux) opérateurs TAX. En outre, nous disposons de plusieurs possibilités d'exprimer un même opérateur. Par exemple, en définissant le *forage vers le haut*, nous avons pu éviter d'employer le *groupement* TAX et ainsi la sémantique de la *jointure*. Un recours à d'autres opérations TAX supplémentaires est très probable dans ce dernier cas. Pour le *forage vers le haut* comme pour le reste des opérateurs XOLAP, nous avons voulu conserver le principe « norme » tout en l'exprimant de la façon la plus simple et la plus efficace.

3.4.2 Conformité entre notre algèbre XOLAP et l'algèbre OLAP classique

Nous avons respecté, quelle que soit la manière de sa mise en œuvre avec TAX, sa définition habituelle dans le contexte OLAP classique. En conséquence, les définitions des opérateurs XOLAP mis en œuvre, indépendamment de TAX, sont les mêmes que celles de leurs équivalents en algèbre OLAP classique. De plus, nous avons donné le résultat de chaque opérateur XOLAP sous forme de cubes XOLAP de même structure que les cubes OLAP. Même les deux opérateurs structuraux, la *transformation d'une dimension en mesure* et *la transformation d'une mesure en dimension* peuvent

être exprimés sous forme de cubes XOLAP car tout arbre TAX est traductible en cube XOLAP, ce qui n'existe dans aucune autre algèbre OLAP.

Tout au long de la présentation de l'algèbre XOLAP, nous avons commencé par la définition générale de chaque opérateur dans le contexte OLAP classique. Notons que les cours que nous avons utilisés ainsi que l'ensemble des articles proposant des algèbres OLAP ne constituent pas des supports complets de l'OLAP. En effet, certains opérateurs sont absents dans certains supports. D'autres sont munis de définitions contradictoires d'un document à un autre, comme le *découpage de cube en tranches* (*slice*) et *détachement d'un dé du cube* (*dice*), qui varient d'un travail à un autre (cf. Section 3.3.3.2). Les présentations des opérateurs OLAP, dans les références utilisées ne présentent pas le principe de leur fonctionnement, tout en se limitant à des définitions simples qui obligent le lecteur à faire un effort supplémentaire pour les comprendre. C'est pourquoi nous avons voulu présenter à travers ce mémoire, en plus de l'algèbre XOLAP, les définitions de chaque opérateur OLAP, et surtout leur mode de fonctionnement, d'une façon simple et technique.

Finalement, comme nous l'avons déjà évoqué (cf. Section 3.4.1), nous avons voulu conserver le principe « norme » de chaque opérateur, même dans le contexte XML. Par exemple, nous avons exprimé le *forage vers le haut* en conservant le principe de l'association *groupement-agrégation*, indispensable à son fonctionnement.

3.5 Mise en pratique de notre algèbre XOLAP

À ce stade, nous avons mis en œuvre, de façon théorique, une algèbre XOLAP complète en unissant les concepts XML et OLAP en un seul. Afin de valider notre approche, nous l'avons implémenté au sein du SGBD natif XML TIMBER.

3.5.1 Recours à TIMBER

Comme nous avons étendu TAX par des opérateurs OLAP, nous avons de mettre en œuvre les requêtes traduisant nos opérateurs XOLAP via le SGBD natif XML TIMBER [PAKC+03]. TIMBER constitue en effet la mise en œuvre de TAX et de ses dérivées logique et physique. En suivant les instructions du manuel d'utilisateur mis en ligne [JPCJ+06], nous avons expérimenté quelques requêtes XML simples. L'interface TIMBER permet à l'utilisateur de saisir, ou de télécharger à partir d'un fichier, sa requête XQuery 1.0 [BCF+07]. L'analyste choisit ensuite un des trois modes d'entrée pour sa requête (XQuery, algèbre logique ou algèbre physique), ainsi que le mode d'affichage du résultat (XML, algèbre logique ou algèbre physique). Finalement, il ne reste qu'à exécuter la requête et examiner son résultat. Pour cela, nous avons conçu une interface Web, détaillée dans la partie suivante.

3.5.2 Mise en pratique de notre algèbre

Dans cette section, nous donnons plus de détails sur l'interface Web qui permet d'exploiter notre algèbre XOLAP en pratique. Pour illustrer cette partie, nous continuons d'utilise notre exemple de données XML multidimensionnelles de ventes. Faute de place, nous ne présentons qu'un exemple d'opérateur de chaque famille: structurel (rotation), ensemblistes (découpage de cube en tranches) et lié à la granularité (forage vers le haut).

3.5.2.1 Principe de l'application

Notre interface permet en premier lieu de sélectionner l'opération XOLAP désirée à partir d'une liste (ou un menu) contenant tous les opérateurs XOLAP de notre algèbre. Ensuite, il peut sélectionner les paramètres caractérisant l'opération pour générer la requête XQuery correspondante. Dans l'étape suivante, cette requête est téléchargée (upload) et exécutée à travers l'interface de TIMBER. Notons toute fois que notre application est strictement indépendante de TIMBER. Nous pouvons la lier à toute autre SGBD natif XML supportant XQuery.

3.5.2.2 Opérateurs implémentés

- **Rotation (rotate)**

Nous savons déjà que la rotation permute les dimensions d'un cube en lui faisant subir une rotation autour d'un de ses axes (dimension). Dans notre algèbre, nous réordonnons les dimensions ciblées par cette opération, autour de l'axe de rotation, dans le fichier XML multidimensionnel final. Pour atteindre ce but pratiquement, l'interface permet à l'analyste de choisir, après la visualisation de son cube, l'axe (la dimension) de rotation. Il ne reste pour l'utilisateur qu'à cliquer sur un bouton « Effectuer une rotation » pour générer la requête XQuery. Un téléchargement suivi d'une exécution de cette requête sur notre entrepôt de données XML avec TIMBER génère le fichier XML de la figure 33.

- **Découpage de cube en tranches (slice)**

Par principe, le *découpage de cube en tranches* détache une ou plusieurs tranches d'un cube suivant un ou plusieurs attributs d'une même dimension. Dans l'interface implémentée, ainsi que dans la requête XQuery générée, nous suivons cette logique.

Grâce à notre interface, l'utilisateur peut choisir les attributs sur lesquels effectuer la coupe. La seule condition que nous imposons aux utilisateurs est que l'ensemble des attributs sélectionnés doit appartenir à une même dimension. La requête XQuery, générée à partir de cette interface sélectionne les faits des fichiers XML multidimensionnels selon les attributs. Le résultat d'une l'exécution de cette opération qui découpe *ventes* en tranches *vente* suivant la *ville* Paris est disponible dans la figure 34.

- **Forage vers le haut (Roll up)**

Un *forage vers le haut* est assuré par une combinaison des opérations de *sélection*, *groupement* multiple, *jointure*, *agrégation* et mise à jour des valeurs de nœuds (*insertion de nœuds*). Cet enchainement d'opérations est traduit dans notre interface comme suit, tout en tenant compte de la requête XQuery que nous voulons générer. Soit un exemple où les dimensions de *groupement* sont *ville* et *produit*, la dimension de *jointure* est *année*, l'*agrégation* employée est la *somme*, et *Temps* représente la valeur du nœud inséré à la place d'*année* en résultat, puisque le *forage vers le haut* est assuré suivant cet axe.

Lors de la mise en œuvre du *forage vers le haut* en pratique, nous nous sommes rendu compte que notre programme est plus souple si la *jointure* est réalisée suivant la dimension qui n'a pas participé au *groupement* (*année* dans notre exemple). Cette solution était citée, mais pas utilisée dans la définition du *forage vers le haut* avec TAX. Pourtant, nous l'avons soutenu dans la pratique pour des buts d'optimisation de notre programme.

L'interface implémentée, disponible en annexe figure 35, facilite la réalisation d'un *forage vers le haut* grâce au protocole suivant.

1- En premier lieu, l'analyste choisit les dimensions de *groupement*. Si nous disposons de *n* dimensions, *n-1* dimensions sont alors les éléments de groupement multiple. Ce sont toutes les dimensions sauf la dimension selon laquelle est assuré le *forage vers le haut* (*ville* et *produit* dans notre exemple).

2- L'analyste sélectionne ensuite sa dimension de jointure, *année* dans notre exemple.

3- Une liste de fonctions d'agrégation est disponible pour choisir celle à utiliser dans le *forage vers le haut*, *somme* dans notre exemple.

4- Enfin, il ne reste pour l'utilisateur qu'à cliquer sur un bouton pour générer la requête permettant d'assurer l'opération XOLAP *forage le haut*.

Dans notre programme, nous associons à la requête générée la fonction XQuery 1.0 de séquence distinct-values( ). Cette dernière joue même rôle que la fonction SQL distinct, en éliminant les valeurs identiques d'un attribut. Malheureusement, TIMBER ne prend pas en charge la fonction

distinct-values( ). Nous avons essayé d'exprimer la requête avec l'algèbre physique, mais c'était impossible, même en ayant recours à l'opérateur *élimination de doublons* (*duplicate-elimination*). Les développeurs de TAX nous ont confirmé ces problèmes. Une modification du code source TIMBER nous a cependant permis d'exécuter la requête équivalente en algèbre physique. Malgré les contraintes liées à TIMBER, notons que notre requête XQuery assure le *forage vers le haut* sur tout autre système supportant XQuery 1.0 avec toutes ses fonctions. Nous avons utilisé l'éditeur XML XMLSpy pour générer le résultat d'un *forage vers le haut* avec XQuery (figure 36).

3.5.2.3 Opérateurs en cours d'implémentation

Nos autres opérateurs sont en cours d'implémentation. Leur mise en œuvre suit le même principe, une interface générant une requête XQuery. Des limitations de XQuery ont toute fois influencé notre travail.

3.5.2.4 Limitations de XQuery rencontrées

Pour implémenter plusieurs de nos opérateurs XOLAP, comme *switch*, *cube et dice*, des problèmes liées aux limitations du langage XQuery nous ont rencontré. Ce dernier touche l'opérateur XOLAP *switch*, précisément, et touche aux autres opérateurs *cube* et *dice* pour l'instant.

Le *changement de position des modalités dans une dimension* (*switch*) permet de permuter deux attributs d'une même dimension au sein d'un cube XOLAP. La requête XQuery correspondante parcourt l'élément représentant la dimension cible un nombre de fois égal au nombre de modalités à interchanger. Elle doit afficher les différentes valeurs de cette même dimension dans l'ordre désiré, qui est également indiqué aussi dans cette requête.

Le problème qui se pose ici est qu'une requête XQuery ne permet pas plus d'un parcours d'un seul élément donné avec une boucle for. Donc, pour réaliser un *changement de position des modalités dans une dimension*, il nous faut combiner un ensemble de résultats de plusieurs requêtes XQuery qui parcoure chacune la dimension cible une fois. Une association des différents résultats de toutes ces requêtes constitue le résultat désiré.

Bien que nous atteignions notre but en suivant cette méthode, elle n'est pas efficace et nous envisageons d'optimiser ce processus dans des travaux futurs.

4 Conclusion et perspectives

Nous avons défini dans ce mémoire le cadre formel d'une nouvelle algèbre XOLAP qui permet d'exécuter des requêtes analytiques OLAP sur des données multidimensionnelles stockées dans des documents XML. Ces travaux se basent sur un état de l'art que nous avons voulu le plus exhaustif possible et qui traite des algèbres OLAP, des algèbres XML et des premiers travaux concernant des approches XML-OLAP. Cette partie de notre mémoire a fait l'objet d'une soumission d'article *survey* à la conférence « Bases de Données Avancées » (BDA) 2007.

Afin de définir notre algèbre XOLAP, nous avons étendu l'algèbre XML TAX par les opérateurs OLAP les plus couramment utilisés et cités dans la littérature : *cube*, *rotate*, *switch*, *roll up*, *drill down*, *slice*, *dice*, *pull* et *push*. De plus, ce travail nous a également amenés à formaliser des notions indispensables, comme les fichiers XML multidimensionnels, les cubes XOLAP et les arbres TAX multidimensionnels.

Finalement, nous avons conçu un prototype logiciel qui met en œuvre nos opérateurs XOLAP et permet de générer le code XQuery correspondant. Cette interface d'interrogation s'appuie actuellement sur le SGBD XML natif TIMBER, qui implémente l'algèbre TAX et ses dérivées, mais elle en est totalement indépendante et pourrait fonctionner avec tout autre SGBD XML supportant XQuery.

Les perspectives qui s'offrent à nous sont de trois ordres. Premièrement, nous comptons poursuivre l'implémentation de notre proposition d'algèbre XOLAP logique au niveau physique, dans le cadre d'une plateforme d'entreposage XML de données complexes en cours de conception dans notre laboratoire [BDD06]. Notre objectif est de permettre, à travers une interface simple et accessible depuis le Web, la construction et la manipulation de cubes XML.

Dans un second temps, et en plus des opérateurs définis, nous comptons incorporer dans notre algèbre et notre prototype de nouveaux opérateurs spécifiques au contexte XML. Il nous paraît, par exemple, particulièrement intéressant de pouvoir effectuer des opérations de forage (*roll up* et *drill down*) sur des hiérarchies de dimensions complexes telles que les *ragged hierarchies* mentionnées par Beyer et al. [BCC+05]. L'objectif de ce travail est de venir en support des extensions en cours du langage XQuery pour les applications décisionnelles.

Pour terminer, l'efficacité du langage XQuery s'étant avérée au cours de nos tests limitée pour exécuter des requêtes décisionnelles, nous envisageons d'exploiter notre algèbre XOLAP comme support de l'optimisation automatique de requêtes OLAP exprimées en XQuery.

Bibliographie

[Adi05] Michel Adiba. *Entrepôts de données Fouille de Données*. Support de cours, Master SI, Université de Grenoble, France, 2004-2005.

[AGS96] Rakesh Agrawal, Ashish Gupta, and Sunita Sarawagi. *Modeling Multidimensional Databases*. Technical report, IBM Almaden Research Center, San Jose, USA, 1996.

[AGS97] Rakesh Agrawal, Ashish Gupta, and Sunita Sarawagi. *Modeling Multidimensional databases*. In 13th International Conference on Data Engineering (ICDE 97), Birmingham, UK, pages 232-243. IEEE Computer Society, 1997.

[BB03] Xavier Baril and Zohra Bellahsène. *Designing and Managing an XML Warehouse*, pages 455-473. XML Data Management: Native XML and XML-enabled Database Systems. Addison Wesley, 2003.

[BC04] Vinayak R. Borkar and Michael J. Carey. *Extending XQuery for Grouping, Duplicate Elimination, and Outer Joins*. In XML 2004, Washington DC, USA, pages 1-11, November 2004.

[BCC+05] Kevin S. Beyer, Donald D. Chamberlin, Latha S. Colby, Fatma Özcan, Hamid Pirahesh, and Yu Xu. *Extending XQuery for Analytics*. In ACM SIGMOD International Conference on Management of Data (SIGMOD 05), Baltimore, USA, pages 503-514. ACM, 2005.

[BCF+07] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. *XQuery 1.0: An XML Query Language*. http://www.w3.org/TR/xquery/. World Wide Web Consortium (W3C), 2007.

[BCH99] David Belanger, Kenneth Ward Church, and Andrew Hume. *Virtual Data Warehousing, Data Publishing, and Call Detail*. In International Workshop on Databases in Telecommunications, Edinburgh, UK, volume 1819 of Lecture Notes in Computer Science, pages 106-117. Springer, 1999.

[BDD06] BDD. *Plateforme d'entreposage XML de données complexes*. Rapport technique, Pôle Bases de Données Décisionnelles (BDD), Laboratoire ERIC, Université Lumière Lyon 2, France, http://bdd.univ-lyon2.fr/?page=projets, 2006.

[Ben07] Fadila Bentayeb. *Systèmes d'Informations Décisionnels*. Support de cours, Master Recherche ECD, Université Lumière Lyon 2, France, 2006-2007.

[BFLC03] Sujoe Bose, Leonidas Fegaras, David Levine, and Vamsi Chaluvadi. *A Query Algebra for Fragmented XML Stream Data*. In 9th International Workshop on Database Programming Languages (DBPL 03), Potsdam, Germany, volume 2921 of Lecture Notes in Computer Science, pages 195-215. Springer, 2003.

[BMCA06] Omar Boussaïd, Riadh Ben Messaoud, Rémy Choquet, and Stéphane Anthoard. *X-Warehousing: An XML-Based Approach for Warehousing Complex Data*. In 10th East European Conference on Advances in Databases and Information Systems (ADBIS 06), Thessaloniki, Greece, volume 4152 of Lecture Notes in Computer Science, pages 39-54. Springer, 2006.

[BNTW95] Peter Buneman, Shamim A. Naqvi, Val Tannen, and Limsoon Wong. *Principles of Programming with Complex Objects and Collection Types*. Theoretical Computer Science, 149(1): 3-48, 1995.

[CAJK01] Damianos Chatziantoniou, Michael O. Akinde, Theodore Johnson, and Samuel Kim. *The MD-join: An Operator for Complex OLAP*. In 17th International Conference on Data Engineering (ICDE 01), Heidelberg, Germany, pages 524-533. IEEE Computer Society, 2001.

[CD99] James Clark and Steve DeRose. *XML Path Language* (*XPath*) *Version 1.0*. http://www.w3.org/TR/xpath. World Wide Web Consortium (W3C), 1999.

[Col90] Latha S. Colby. *A recursive algebra for nested relations*. Information Systems, 15(5): 567-582, 1990.

[CRF00] Donald D. Chamberlin, Jonathan Robie, and Daniela Florescu. *Quilt: An XML Query Language for Heterogeneous Data Sources*. In 3rd International Workshop on The World Wide Web and Databases (WebDB 00), Dallas, USA, volume 1997 of Lecture Notes in Computer Science, pages 1-25. Springer, 2000.

[CT97] Luca Cabibbo and Riccardo Torlone. *Querying multidimensional databases*. In 6th International Workshop of Database Programming Languages (DBPL 97), Estes Park, USA, volume 1369 of Lecture Notes in Computer Science, pages 319-335. Springer, 1997.

[CT98] Luca Cabibbo and Riccardo Torlone. *A logical approach to multidimensional databases*. In 6th International Conference on Extending Database Technology (EDBT 98), Valencia, Spain, volume 1377 of Lecture Notes in Computer Science, pages 183-197. Springer, 1998.

[DBRA05] Jérôme Darmont, Omar Boussaïd, Jean-Christian Ralaivao, and Kamel Aouiche. *An Architecture Framework for Complex Data Warehouses*. In 7th International Conference on Enterprise Information Systems (ICEIS 05), Miami, USA, pages 370-373, 2005.

[DD97] C.J. Date and Hugh Darwen. *A Guide to SQL Standard*, 4th Edition. Addison-Wesley, 1997.

[DFF+98] Alin Deutsch, Mary Fernández, Daniela Florescu, Alon Levy, and Dan Suciu. *XML-QL: A Query Language for XML*. http://www.w3.org/TR/NOTE-xml-ql/. World Wide Web Consortium (W3C), 1998.

[DT99] Anindya Datta and Helen Thomas. *The cube data model: a conceptual model and algebra for on-line analytical processing in data warehouses*. Decision Support Systems, 27(3): 289-301, 1999.

[FHP02] Flavius Frasincar, Geert-Jan Houben, and Cristian Pau. *XAL: an Algebra for XML Query Optimization*. In 13th Australasian Database Conference (ADC 02), Melbourne, Australia, volume 5 of CRPIT. Australian Computer Society, 2002.

[FLBC02] Leonidas Fegaras, David Levine, Sujoe Bose, and Vamsi Chaluvadi. *Query processing of streamed XML data*. In 11th International Conference on Information and Knowledge Management (CIKM 02), McLean, USA, pages 126-133. ACM, 2002.

[FSW00] Mary F. Fernández, Jérôme Siméon, and Philip Wadler. *An Algebra for XML Query*. In 20th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 00), New Delhi, India, volume 1974 of Lecture Notes in Computer Science, pages 11-45. Springer, 2000.

[Gar07] Georges Gardarin. *Data warehouse*. Support de cours, Université de Versailles Saint-Quentin-en-Yvelines, France, 2007.

[GBLP96] Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. *Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Roll-Up*. In 12th International Conference on Data Engineering (ICDE 96), New Orleans, USA, pages 152-159. IEEE Computer Society, 1996.

[GCB+97] Jim Gray, Surajit Chaudhuri, Adam Bosworth,Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. *Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals*. Journal of Data Mining and Knowledge Discovery, 1(1): 29-53, 1997.

[GL97] Marc Gyssens and Laks V. S. Lakshmanan. *A foundation for multi-dimensional databases*. In 23rd International Conference on Very Large Data Bases (VLDB 97), Athens, Greece, pages 106-115. Morgan Kaufmann, 1997.

[GLS96] Marc Gyssens, Laks V. S. Lakshmanan, and Iyer N. Subramanian. *Tables as a paradigm for querying and restructuring*. In 15th ACM SIGACT-SIGMODSIGART Symposium on Principles of Database Systems (PODS 96), Montreal, Canada, pages 93-103. ACM, 1996.

[GVD+01] Leonidas Galanis, Efstratios Viglas, David J. DeWitt, Jeffrey F. Naughton, and David Maier. *Following the paths of XML Data: An algebraic framework for XML query evaluation*. Technical report, University of Wisconsin, USA, 2001.

[Hac99] Richard D. Hackathorn. *Web farming for the data warehouse*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 1999.

[HBH03] Wolfgang Hümmer, Andreas Bauer, and Gunnar Harde. *XCube: XML for data warehouses*. In ACM 6th International Workshop on Data Warehousing and OLAP (DOLAP 03), New Orleans, USA, pages 33-40, 2003.

[HWW06] Philippe Le Hégaret, Ray Whitmer, and Lauren Wood. *Document Object Model (DOM)*. http://www.w3.org/DOM/. World Wide Web Consortium (W3C), 2006.

[JLST01] H. V. Jagadish, Laks V. S. Lakshmanan, Divesh Srivastava, and Keith Thompson. *TAX: A Tree Algebra for XML*. In 8th International Workshop on Database Programming Languages (DBPL 01), Frascati, Italy, volume 2397 of Lecture Notes in Computer Science, pages 149-164. Springer, 2001.

[JMP01] Mikael R. Jensen, Thomas H. Moller, and Torben B. Pedersen. *Specifying OLAP cubes on XML data*. Journal of Intelligent Information Systems, 17(2-3): 255-280, 2001.

[JPCJ+06] H. V. Jagadish, Jignesh M. Patel, Adriane Chapman, Magesh Jayapandian, Yunyao Li, Stelios Paparizos, Nuwee Wiwatwattana, Cong Yu, Shurug Al-Khalifa, Andrew Nierman, Keith Thompson, Yuqing Wu, Yi Yu, Nick Koudas, Laks V. S. Lakshmanan, and Divesh Srivastava. *TIMBER*. http://www.eecs.umich.edu/db/timber/. University of Michigan, USA, 2006.

[Kay06] Michael Kay. *Positional Grouping in XQuery*. In 3rd International Workshop on XQuery Implementation, Experience and Perspectives (XIME-P 06), Chicago, USA, June 2006.

[Law06] George Lawton. *Making business intelligence more useful*. Computer, 39(9): 14-16, 2006.

[LMW96] Leonid Libkin, Rona Machlin, and Limsoon Wong. *A query language for multidimensional arrays: Design, implementation, and optimization techniques.* In ACM SIGMOD International Conference on Management of Data (SIGMOD 96), Montreal, Canada, pages 228-239. ACM, 1996.

[LW96] Chang Li and Xiaoyang Sean Wang. *A data model for supporting on-line analytical processing*. In 5th International Conference on Information and Knowledge Management (CIKM 96), Rockville, USA, pages 81-88. ACM, 1996.

[LW97] Leonid Libkin and Limsoon Wong. *Query Languages for Bags and Aggregate Functions*. Journal of Computer and System Sciences, 55(2): 241-272, 1997.

[NNNT02] Tapio Niemi, Marko Niinimäki, Jyrki Nummenmaa, and Peter Thanisch. *Constructing an OLAP cube from distributed XML data*. In 5th International Workshop on Data Warehousing and OLAP (DOLAP 02), McLean, USA, pages 22-27. ACM, 2002.

[NZ06] Leonid Novak and Alexandre V. Zamulin. *An XML Algebra for XQuery*. In 10th East European Conference on Advances in Databases and Information Systems (ADBIS 06), Thessaloniki, Greece, volume 4152 of Lecture Notes in Computer Science, pages

4-21. Springer, 2006.

[OMG07] OMG. *Unified Modeling Language*. http://www.uml.org. Object Management Group (OMG), 2007.

[PAKC+03] Stelios Paparizos, Shurug Al-Khalifa, Adriane Chapman, H. V. Jagadish, Laks V. S. Lakshmanan, Andrew Nierman, Jignesh M. Patel, Divesh Srivastava, Nuwee Wiwatwattana, Yuqing Wu, and Cong Yu. *TIMBER: A Native System for Querying XML*. In 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD 03), San Diego, USA, page 672. ACM, 2003.

[PAKJ+02] Stelios Paparizos, Shurug Al-Khalifa, H.V. Jagadish, Andrew Nierman, and Yuqing Wu. *A physical algebra for XML*. Technical report, University of Michigan, USA, 2002.

[PHS05] Byung-Kwon Park, Hyoil Han, and Il-Yeol Song. *XML-OLAP: A Multidimensional Analysis Framework for XML Warehouses*. In 7th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 05), Copenhagen, Denmark, volume 3589 of Lecture Notes in Computer Science, pages 32-42. Springer, 2005.

[Pok02] Jaroslav Pokorný. *XML Data Warehouse: Modelling and Querying*. In 5th International Baltic Conference (BalticDB&IS 02), Tallin, Estonia, pages 267-280. Institute of Cybernetics at Tallin Technical University, 2002.

[PP03] Dennis Pedersen and Torben Bach Pedersen. *Achieving adaptivity for OLAP-XML federations*. In 6th International Workshop on Data Warehousing and OLAP (DOLAP 03), New Orleans, USA, pages 25-32. ACM, 2003.

[PPP04] Dennis Pedersen, Jesper Pedersen, and Torben Bach Pedersen. *Integrating XML Data in the TARGIT OLAP System*. In 20th International Conference on Data Engineering (ICDE 2004), Boston, USA, pages 778-781. IEEE Computer Society, 2004.

[Pra06] Sujeet Pradhan. *An Algebraic Query Model for Effective and Efficient Retrieval of XML Fragments*. In 32nd International Conference on Very Large Data Bases (VLDB 06), Seoul, Korea, pages 295-306. ACM, 2006.

[PRP02a] Dennis Pedersen, Karsten Riis, and Torben Bach Pedersen. *A Powerful and SQL-Compatible Data Model and Query Language for OLAP*. In 13th Australasian Database Conference (ADC 02), Melbourne, Australia. Australian Computer Society, 2002.

[PRP02b] Dennis Pedersen, Karsten Riis, and Torben Bach Pedersen. *Query optimization for OLAP-XML federations*. In 5th International Workshop on Data Warehousing and OLAP (DOLAP 02), McLean, USA, pages 57-64. ACM, 2002.

[PWLJ04] Stelios Paparizos, Yuqing Wu, Laks V. S. Lakshmanan, and H. V. Jagadish. *Tree Logical Classes for Efficient Evaluation of XQuery*. In SIGMOD International Conference on Management of Data (SIGMOD 04), Paris, France, pages 71-82. ACM, 2004.

[Qui06] Liam Quin. *Extensible Markup Language (XML)*. http://www.w3.org/XML/. World Wide Web Consortium (W3C), 2006.

[RTZ01] Franck Ravat, Olivier Teste, and Gilles Zurfluh. *Modélisation multidimensionnelle des systèmes décisionnels*. In 1ères Journées francophones Extraction et Gestion des Connaissances (EGC 01), Nantes, France, pages 201-212. Hermès, 2001.

[RTZ06a] Franck Ravat, Olivier Teste, and Gilles Zurfluh. *Algèbre OLAP et langage graphique*. In 24ème Congrès en Informatique des Organisation et Systèmes d'Information de Décision (INFORSID 06), Hammamet, Tunisie, pages 1039-1054, 2006.

[RTZ06b] Franck Ravat, Olivier Teste, and Gilles Zurfluh. *Constraint-Based Multi-Dimensional Databases*, pages 323-368. Database Modeling for Industrial Data Management. Idea Group Publishing, 2006.

[Sho97] Arie Shoshani. *OLAP and Statistical Databases: Similarities and Differences*. In 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 97), Tucson, USA, pages 185-196. ACM, 1997.

[TD01] Helen Thomas and Anindya Datta. *A Conceptual Model and Algebra for On-Line Analytical Processing in Decision Support Databases*. Information Systems Research, 12(1): 83-102, 2001.

[Van05] Christelle Vangenot. *Datawarehouse*. Support de cours, Département d'informatique, Laboratoire de bases de données, Ecole polytechnique fédérale de Lausanne, Suisse, 2005.

[VBR03] Boris Vrdoljak, Marko Banek, and Stefano Rizzi. *Designing Web Warehouses from XML Schemas*. In 5th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 03), Prague, Czech Republic, volume 2737 of Lecture Notes in Computer Science, pages 89-98. Springer, 2003.

[Ver06] Boris Verhaegen. *Requêtes OLAP sur une base de données XML native*. Master's thesis, Université Libre de Bruxelles, Belgique, 2006.

[Wik07] Wikipédia. *Hypercube OLAP*. http://fr.wikipedia.org/wiki/Hypercube_OLAP. Wikipédia, 2007.

[ZDW+03] Xin Zhang, Katica Dimitrova, Ling Wang, Maged El-Sayed, Brian Murphy, Bradford Pielech, Mukesh Mulchandani, Luping Ding, and Elke A. Rundensteiner. *Rainbow: Multi-XQuery Optimization Using Materialized XML Views*. In 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD 03), San Diego, USA, page 671. ACM, 2003.

[ZPR02] Xin Zhang, Bradford Pielech, and Elke A. Rundesnteiner. *Honey, I shrunk the XQuery!: an XML algebra optimization approach*. In 4th International Workshop on Web Information and Data Management (WIDM 02), McLean, USA, pages 15-22. ACM, 2002.

[ZWLZ05] Ji Zhang, Wei Wang, Han Liu, and Sheng Zhang. *X-warehouse: building query pattern-driven data*. In 14th International Conference on World Wide Web (WWW 05), Chiba, Japan, pages 896-897. ACM, 2005.

[ZY04] M. Zhang and J.T. Yao. *XML Algebras for Data Mining*. In Data Mining and Knowledge Discovery: Theory, Tools, and Technology VI, Orlando, USA, volume 5433 of SPIE proceedings series, pages 209-217. SPIE, 2004.

Annexes

```
<vente>
   <ville>Paris</ville>
   <produit>Souris</produit>
   <annee>2004</annee>
   <prix>70</prix>
</vente>
```

Figure 1: Fait XML vente de l'entrepôt ventes

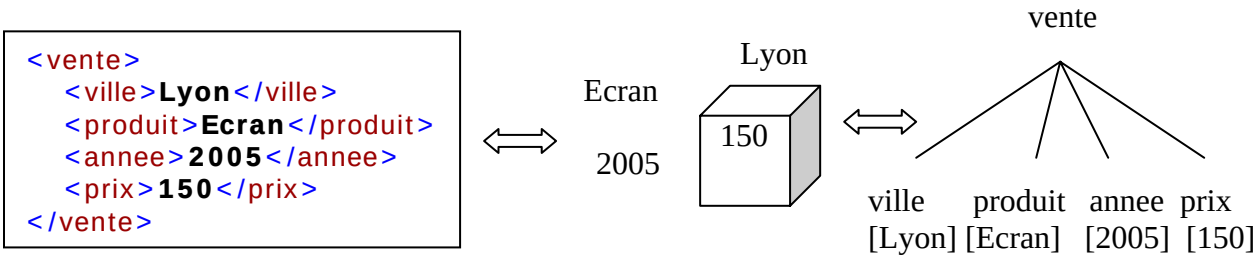


Figure 2 : Fait XML vente sous ses trois présentations : fichier XML multidimensionnel, cellule XOLAP et arbre TAX multidimensionnel

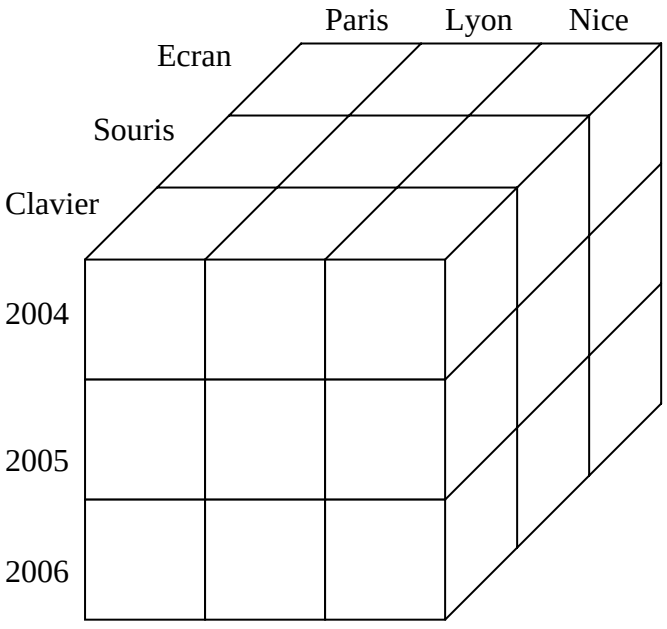


Figure 3 : Structure du cube XOLAP représentant l'entrepôt de données XML ventes

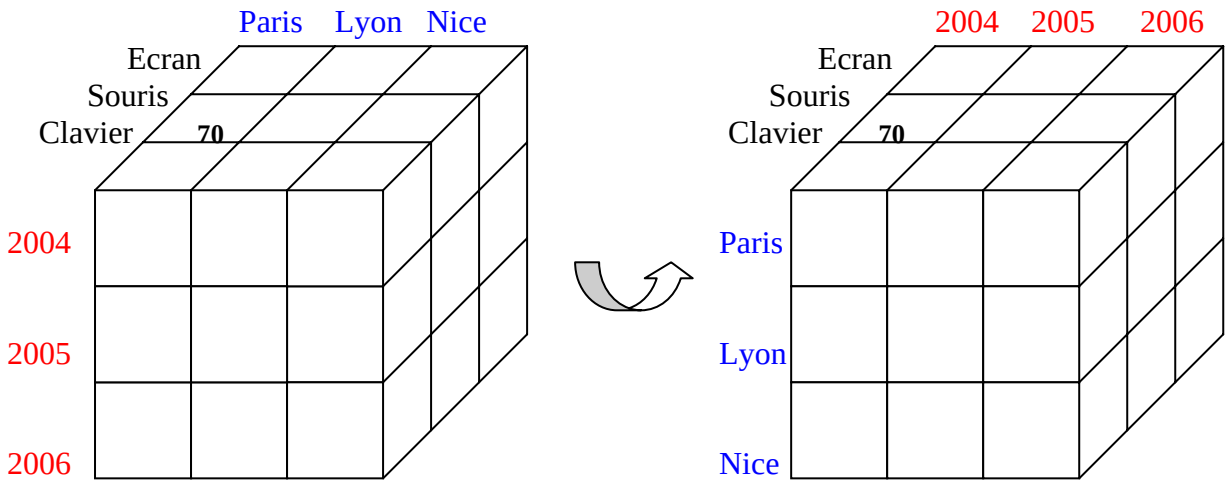


Figure 4 : Rotation du cube XOLAP ventes suivant la dimension produit

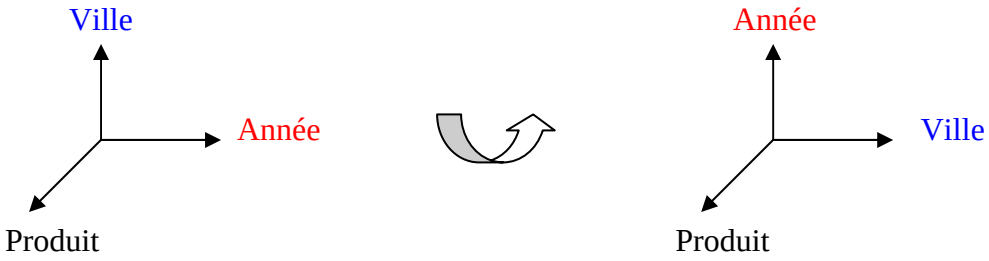


Figure 5 : Rotation des axes représentant les dimensions de l'entrepôt XML ventes

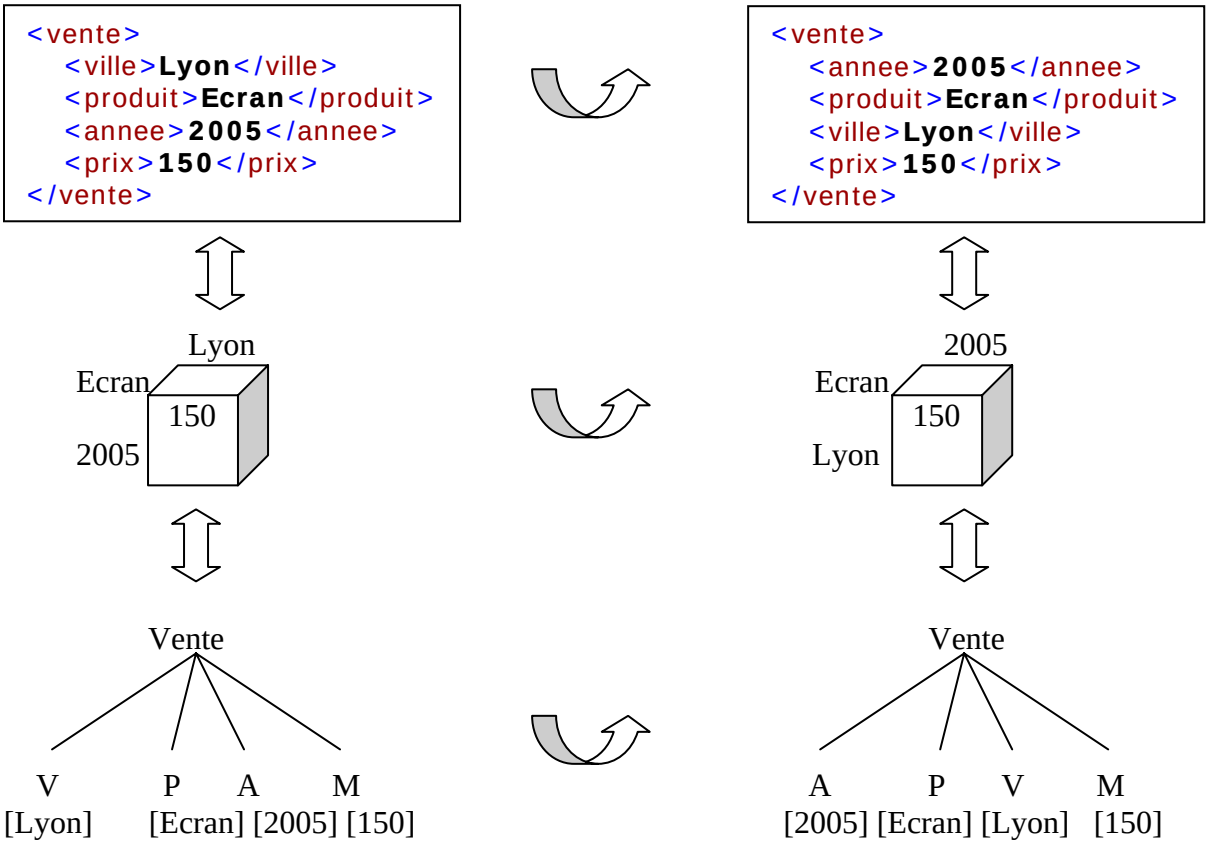


Figure 6 : Exemple d'une rotation d'un fait XML sous les trois formes :
fichier XML multidimensionnel, cellule XOLAP et arbre TAX multidimensionnel

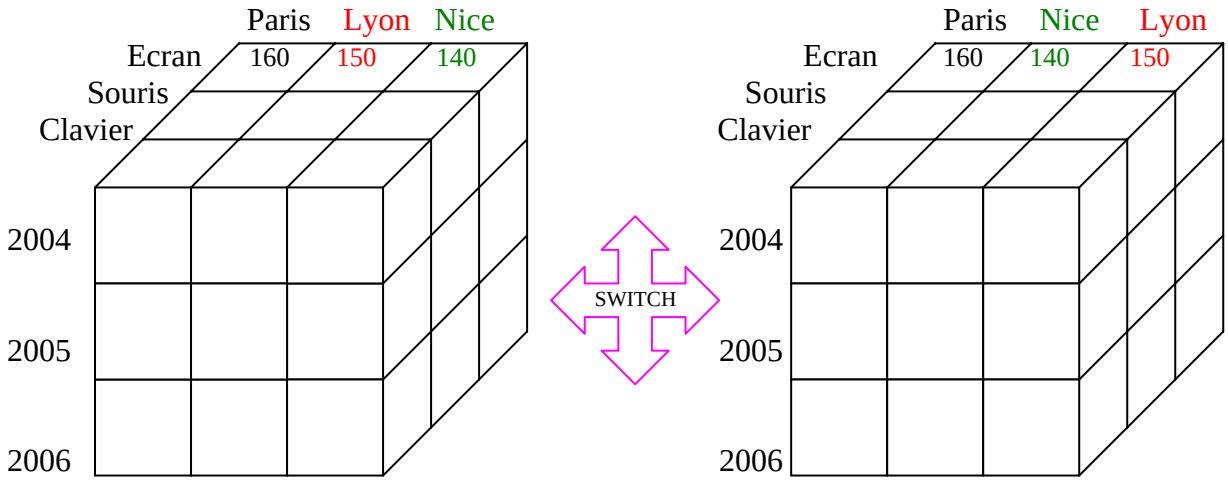


Figure 7 : Changement de position des modalités dans une dimension dans le cube XOLAP ventes

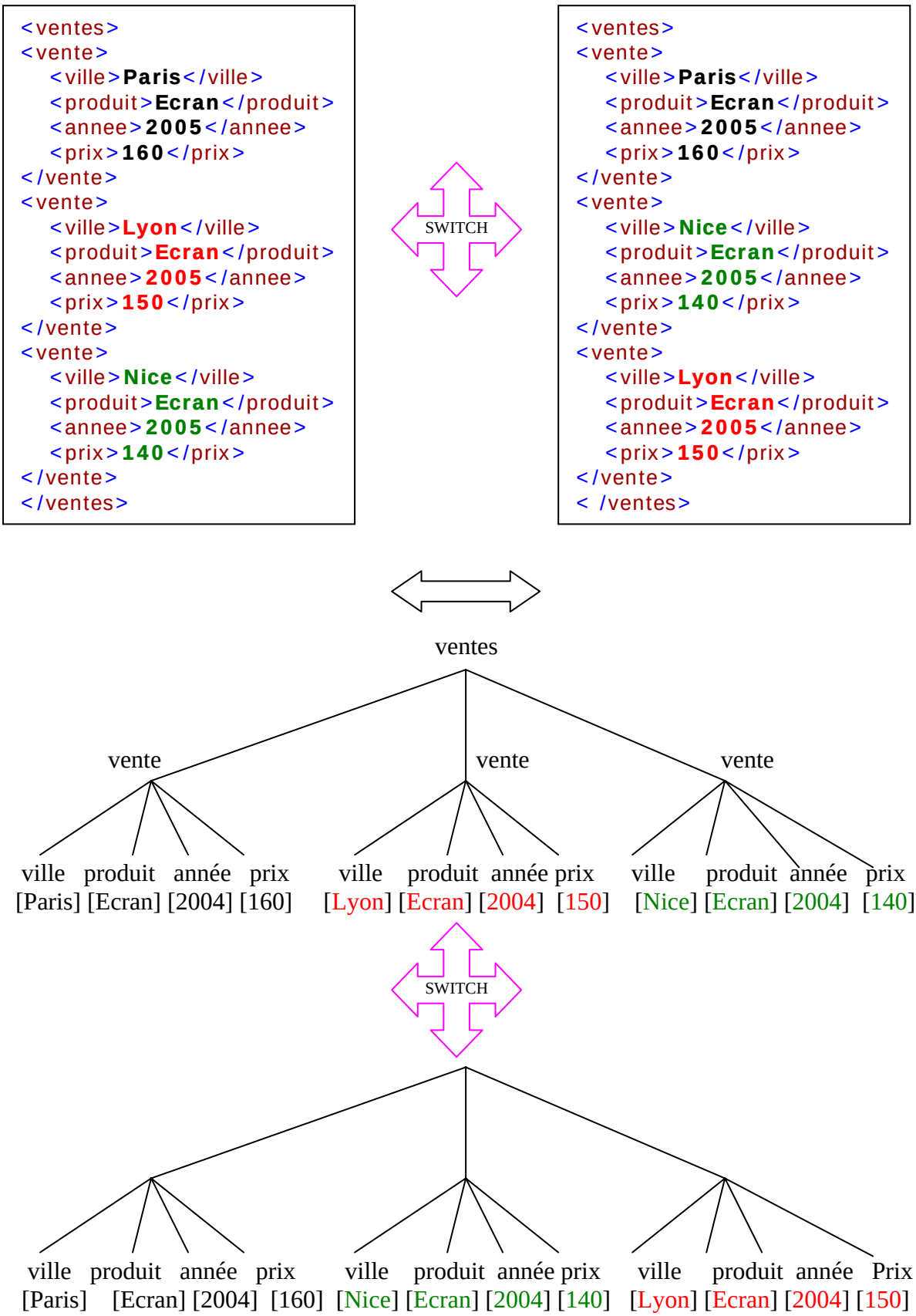
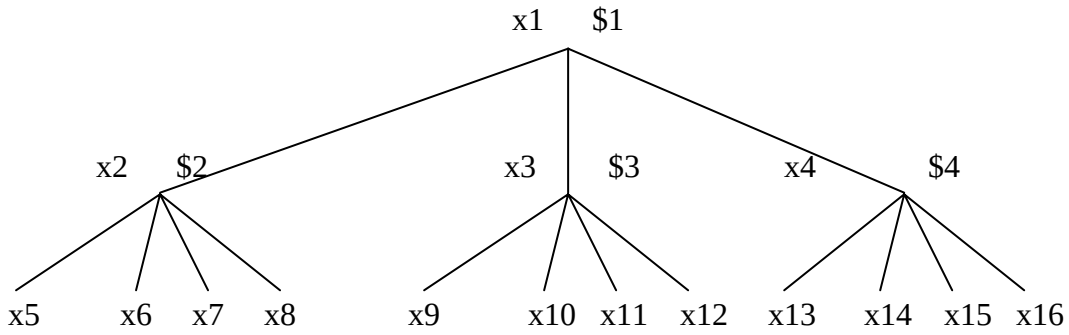


Figure 8 : Changement de position des modalités dans une dimension sous les trois formes, fichier XML multidimensionnel, cellule XOLAP et arbre TAX multidimensionnel



Arbre modèle

$1, $2, $3, $4

Liste de réordonnancement

f(x1)= ventes
f(x2) = f(x3) = f(x4) = vente
f(x5)=Paris
f(x9)=Lyon
f(x13)=Nice

Différents valeurs que prend la fonction de valeur d'arbre

Figure 9 : Éléments en entrée du réordonnancement TAX pour réaliser un changement de positions des modalités dans une dimension

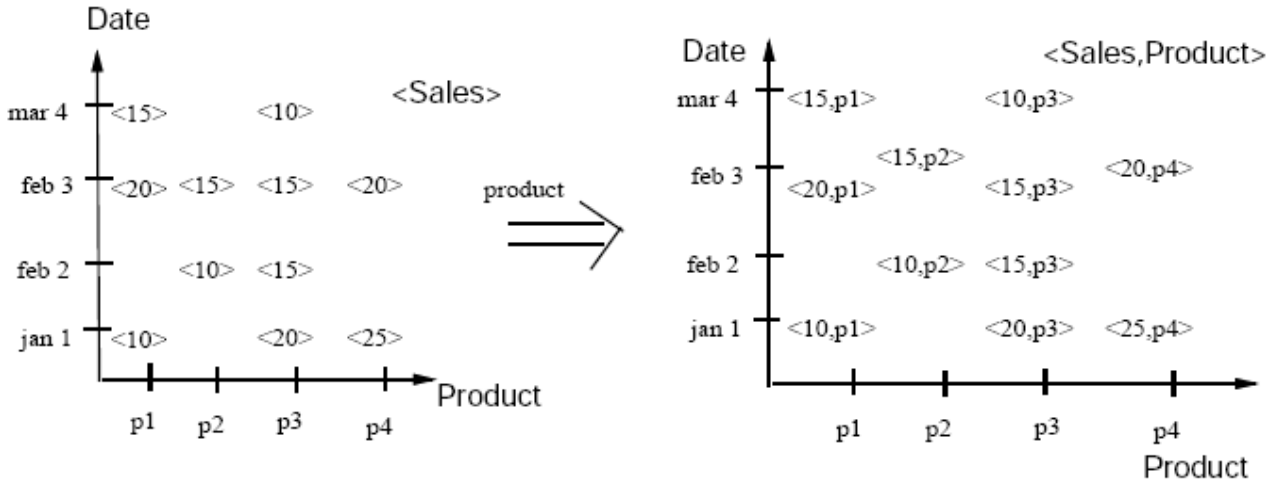


Figure 10 : Transformation d'une mesure en dimension dans l'algèbre d'Agrawal et al. [AGS96, AGS97]

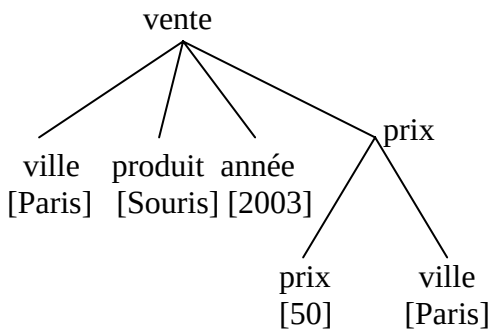
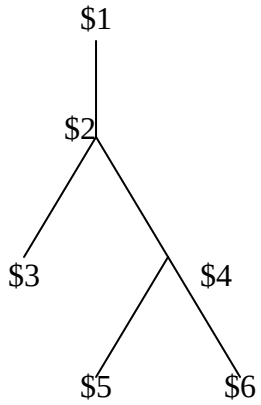


Figure 11 : Transformation d'une mesure en dimension avec TAX



$1.tag=ventes
$2.tag=vente
$3.tag=$5.tag=ville
$3.content=$5.content
$4.tag=$6.tag =prix
$4.content=$6.content

Figure 12 : Modèle d'arbre pour l'opération TAX copier-et-coller

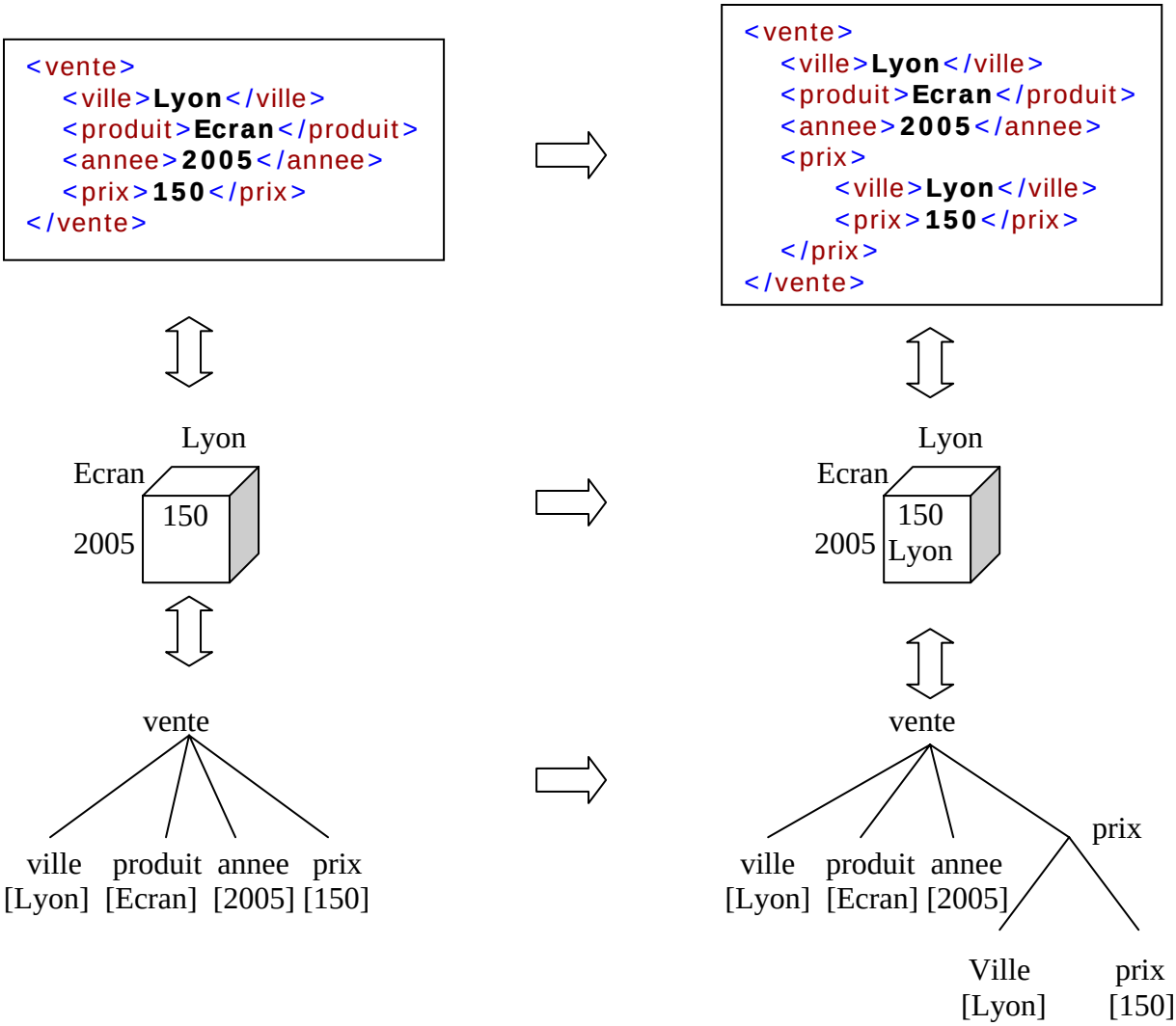


Figure 13 : Transformation d'une mesure en dimension avec toutes les représentations multidimensionnelles (fichier XML, cellule XOLAP et arbre TAX)

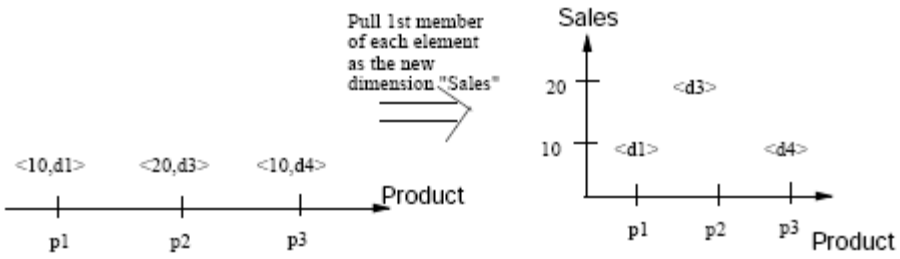


Figure 14 : Transformation d'une dimension en mesure dans l'algèbre d'Agrawal et al. [AGS96, AGS97]

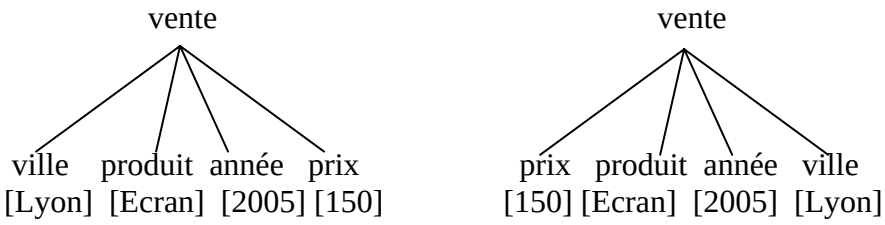


Figure 15 : Transformation d'une dimension en mesure avec TAX

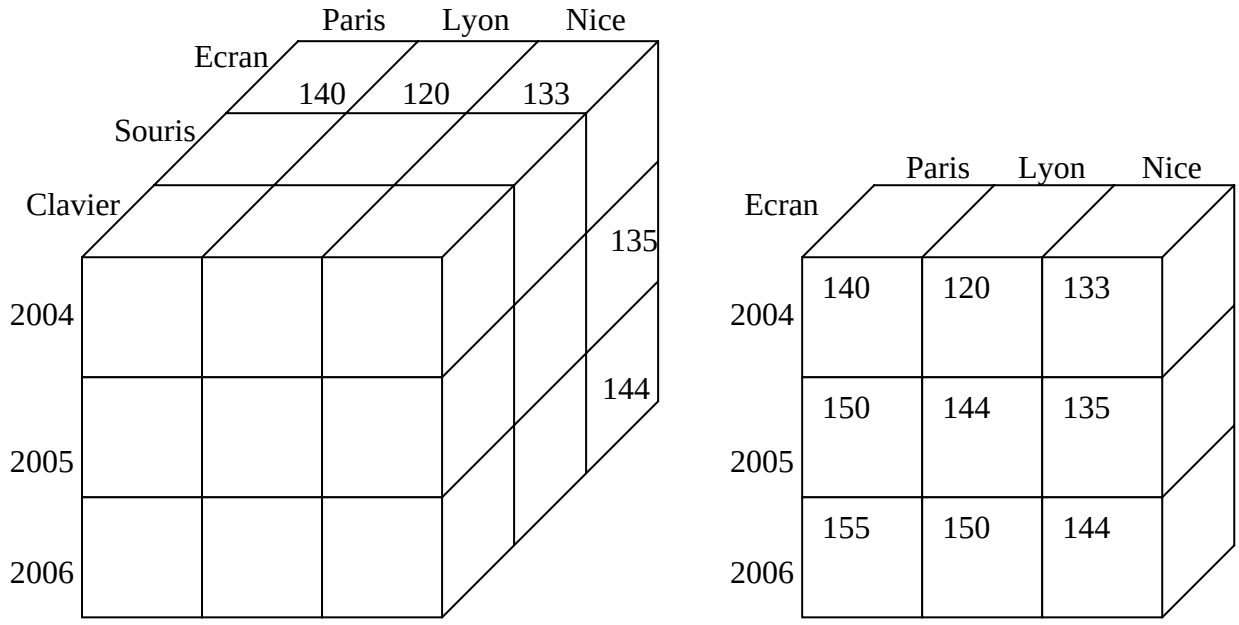


Figure 16 : Découpage de cube XOLAP ventes en tranches suivant le produit Ecran

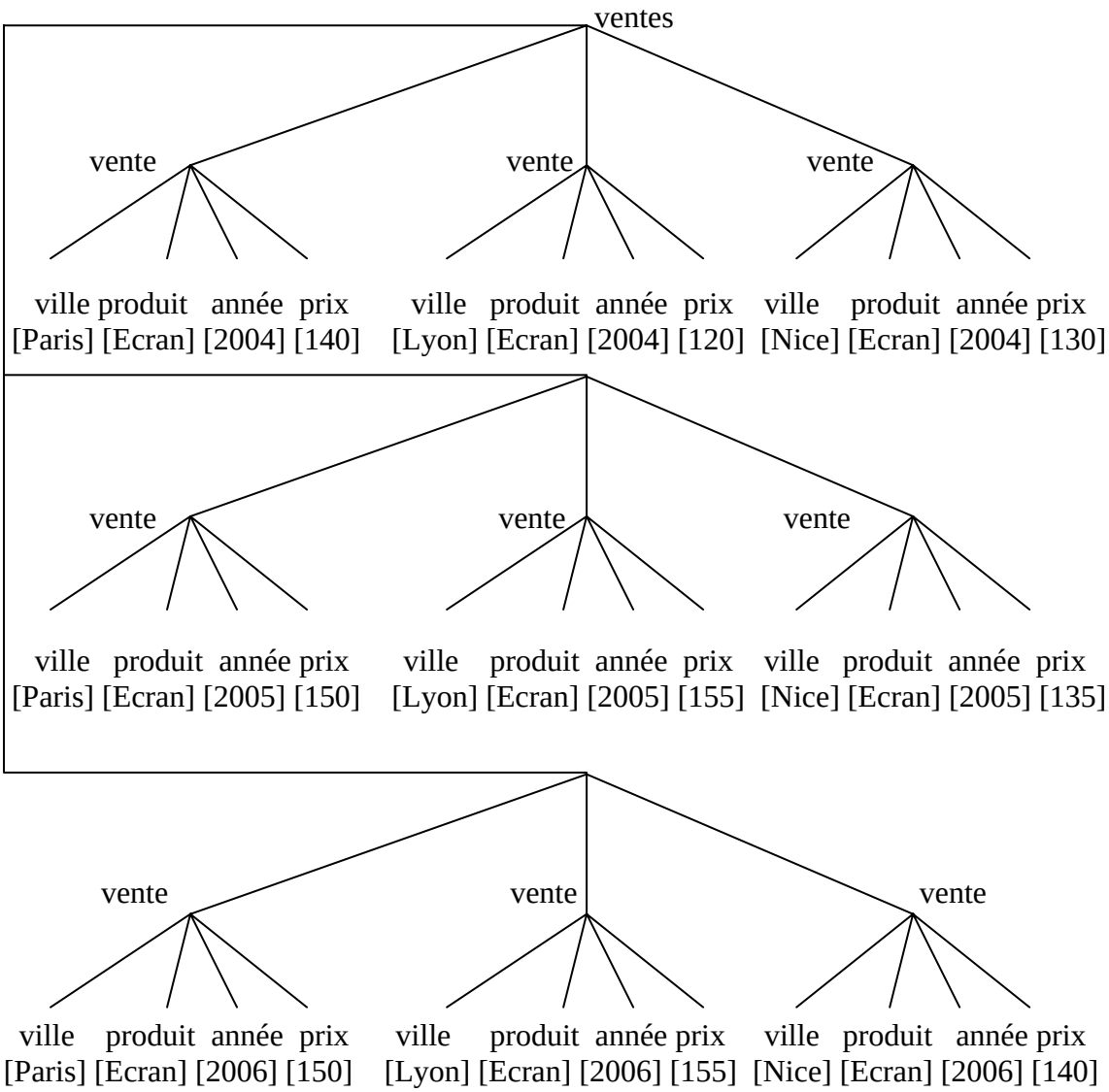
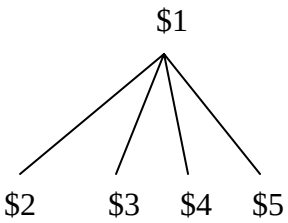


Figure 17 : Découpage de l'entrepôt en tranches avec TAX



avec : $1.tag = vente &
$2.tag = ville &
$3.tag = produit &
$3.content = "Ecran" &
$4.tag = année &
$5.tag = prix

Figure 18 : Modèle d'arbre pour une sélection TAX découpant l'arbre en tranches

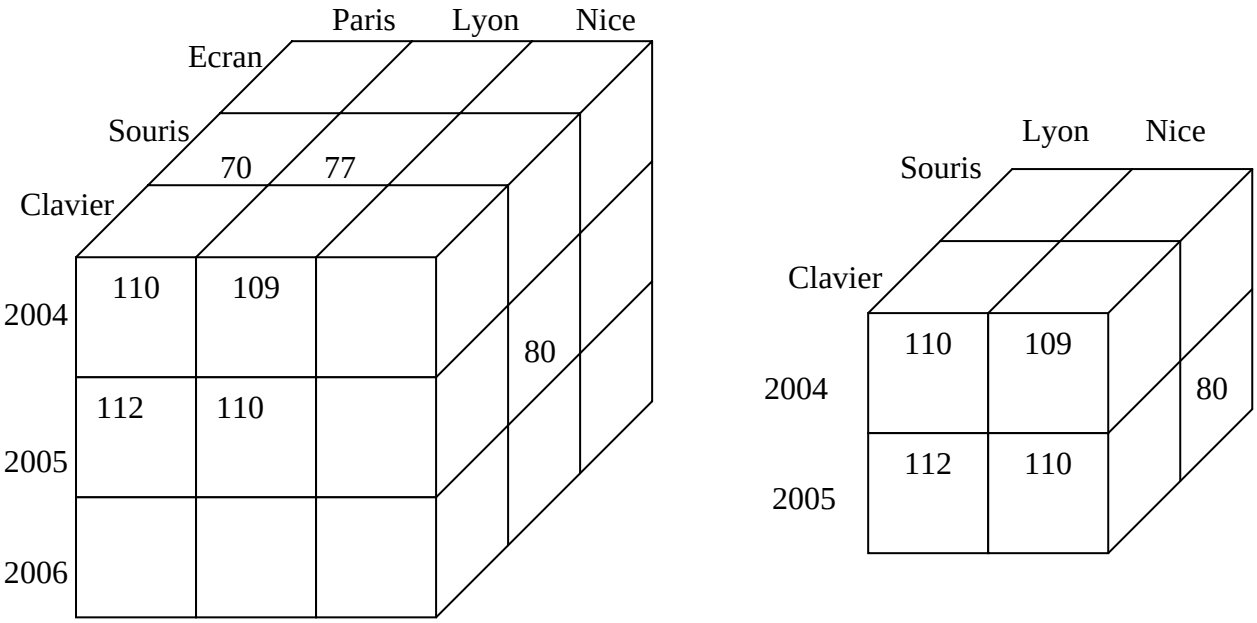
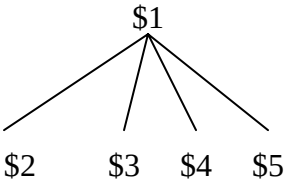


FIGURE 19 : Détachement d'un dé du cube XOLAP ventes




```
$1.tag = vente &
$2.tag = ville &
$2.content = "Lyon" OR "Nice" &
$3.tag = produit &
$3.content = "Souris" OR "Clavier" &
$4.tag = année &
$4.content = "2004" OR "2005" &
$5.tag = prix
```

Figure 20 : Modèle d'arbre pour une sélection TAX découpant l'arbre en tranches

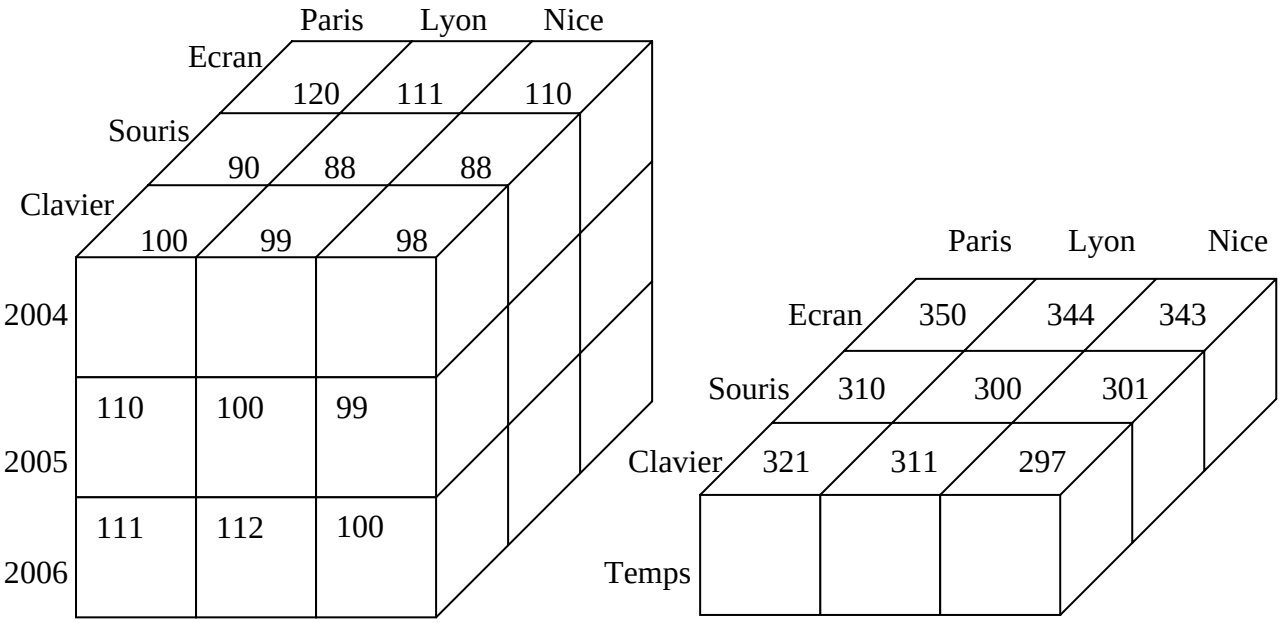


Figure 21 : Forage vers le haut suivant l'axe année (ou Temps) sur le cube XOLAP ventes -La fonction d'agrégation utilisée ici est somme-.

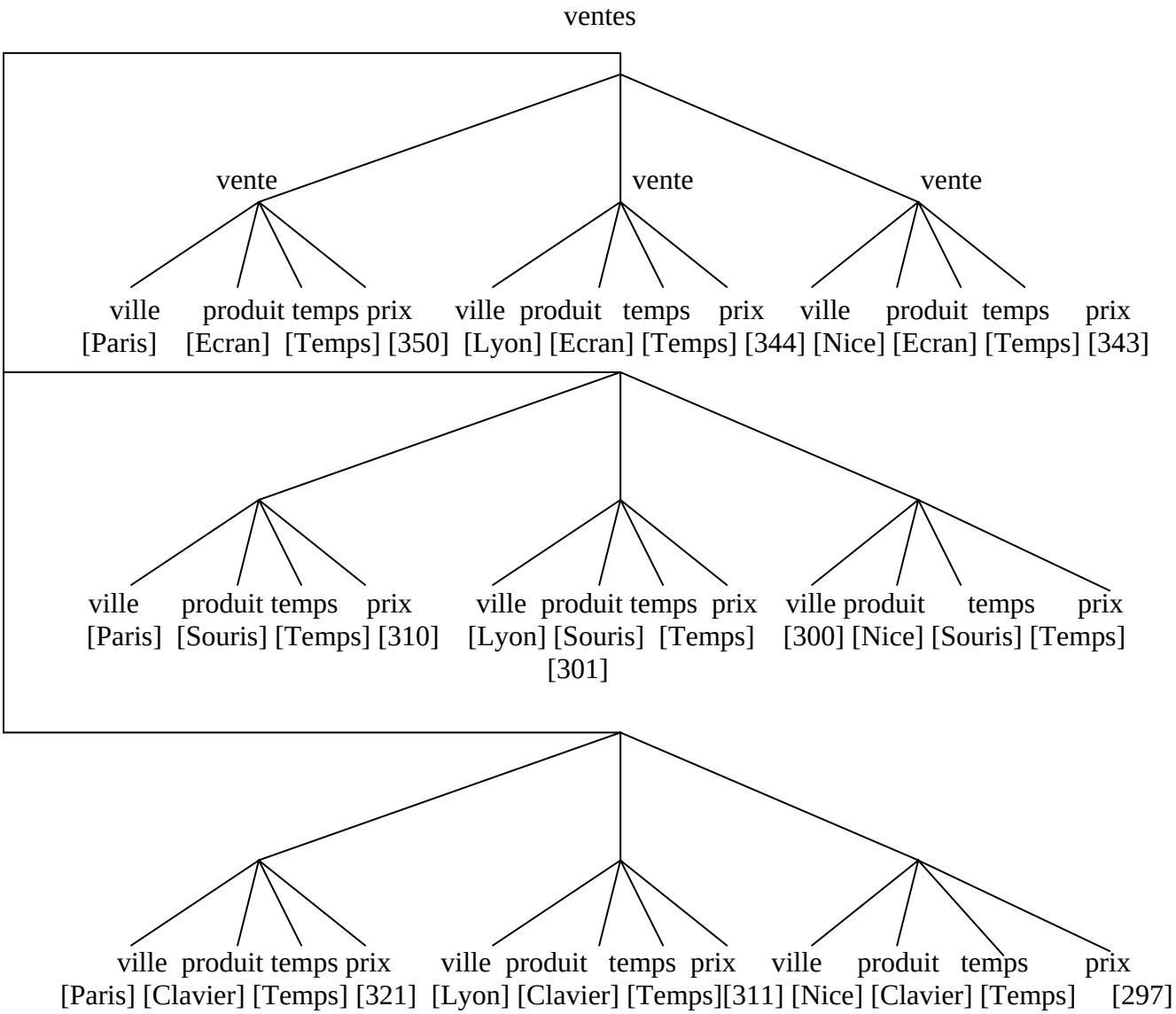
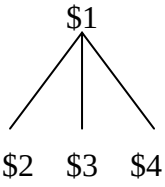


Figure 22 : Forage vers le haut avec TAX



$1.tag = vente
$2.tag = ville
$3.tag = produit
$4.tag = prix

Figure 23 : Modèle pour le groupement utilisé dans le forage le haut TAX

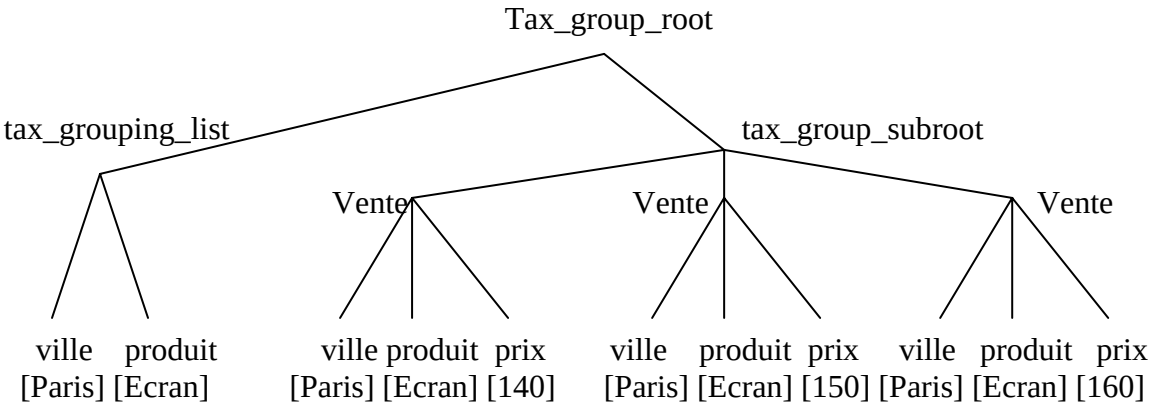
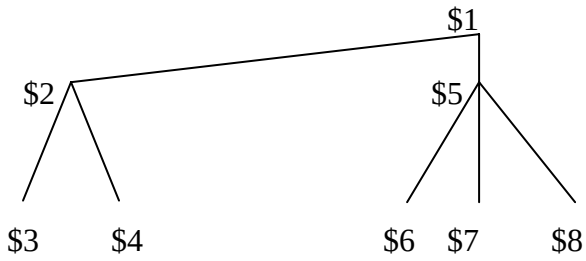


Figure 24 : Groupement selon la ville Paris et le produit Ecran dans le forage vers le haut TAX



$5.tag=vente
$3.content = $6.content
$4.content = $7.content
$3.tag = $6.tag = V
$4.tag = $7.tag = P
$8.tag = M

Figure 25 : Modèle pour la jointure utilisé dans le forage vers le haut TAX

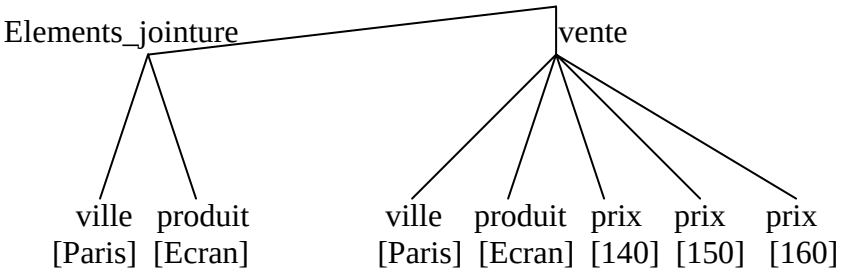


Figure 26 : Jointure selon la ville Paris et le produit Ecran pour le forage vers le haut TAX

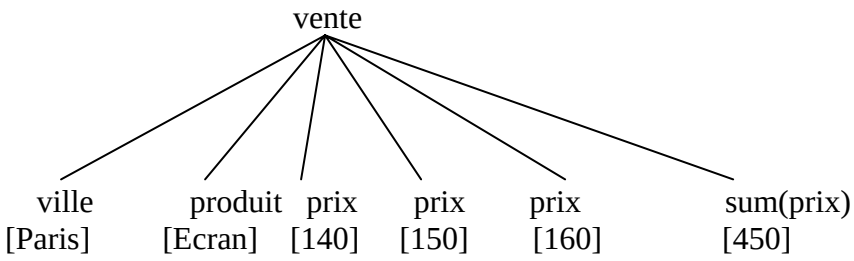


Figure 27 : Calcul d'agrégation pour le forage vers le haut TAX

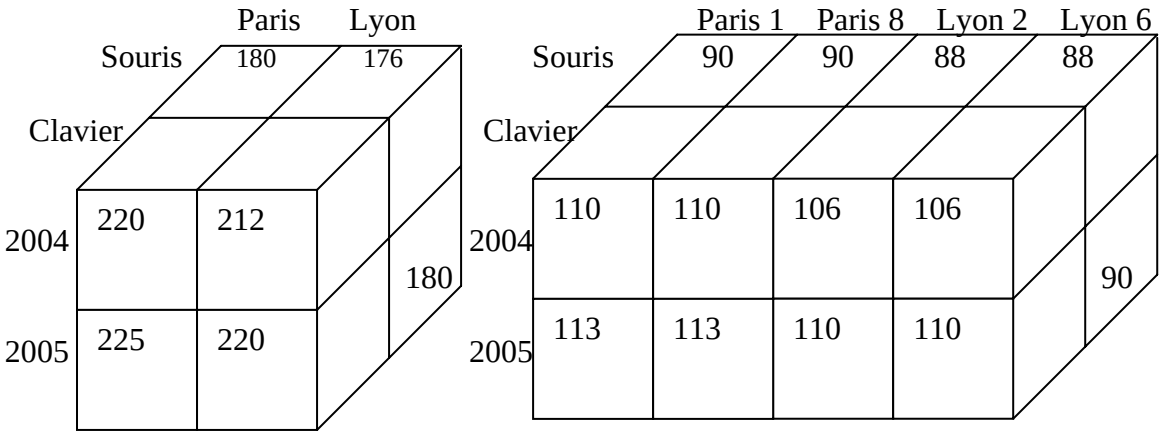


Figure 28 : Forage vers le bas sur le cube XOLAP ventes

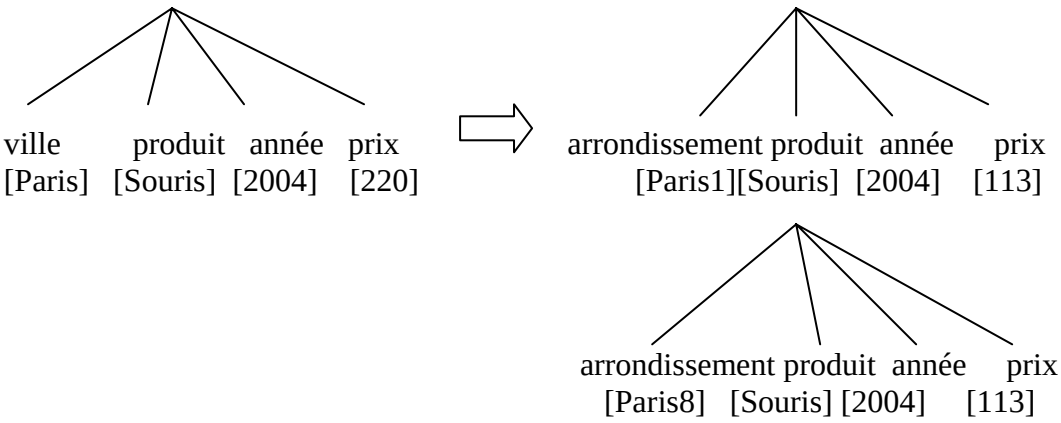
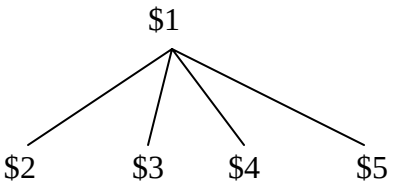
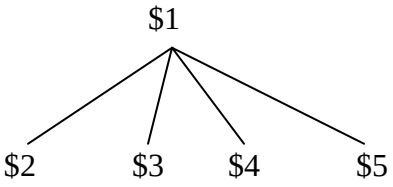


Figure 29 : Forage vers le bas avec TAX



$1.tag=vente &
$2.tag=ville &
$2.content= Paris **OR** $2.content= Lyon &
$3.tag=produit &
$4.tag=année &
$5.tag=prix

Figure 30 : Modèle pour la suppression des arbres supplémentaires dans le forage vers le bas TAX

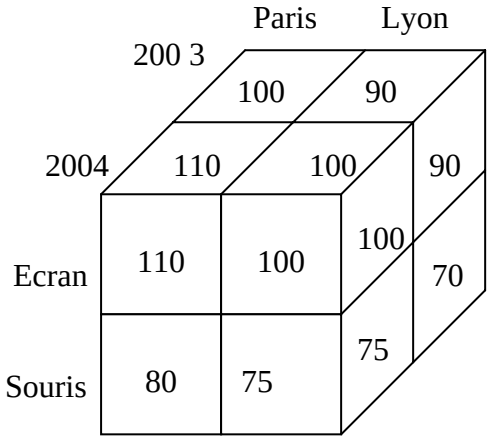


$1.tag=vente &
$2.tag=arrondissement &
$3.tag=produit &
$4.tag=année &
$5.tag=prix

Arbre modèle

Liste de copie = {(vente, Paris1, Souris, 2004, 90) ,
(vente, Paris8, Souris, 2004, 90) ,
(vente, Lyon2, Souris, 2004, 88) ,
(vente, Lyon6, Souris, 2004, 88) ,
(vente, Paris1, Clavier, 2004, 110) ,
(vente, Paris8, Clavier, 2004, 110) ,
(vente, Lyon2, Clavier, 2004, 106) ,
(vente, Lyon6, Clavier, 2004, 106) ,
(vente, Paris1, Souris, 2005, 100) ,
(vente, Paris8, Souris, 2005, 100) ,
(vente, Lyon2, Souris, 2005, 90) ,
(vente, Lyon6, Souris, 2005, 90) ,
(vente, Paris1, Clavier, 2005, 113) ,
(vente, Paris8, Clavier, 2005, 113) ,
(vente, Lyon2, Clavier, 2005, 110) ,
(vente, Lyon6, Clavier, 2005, 110)}

Figure 31 : Eléments en entrée de copier-et-coller dans le forage vers le bas TAX



Notre cube de données se compose de huit cellules, donc de huit sous-arbres TAX (de *v1* à *v8*) réunis sous la même racine ventes.

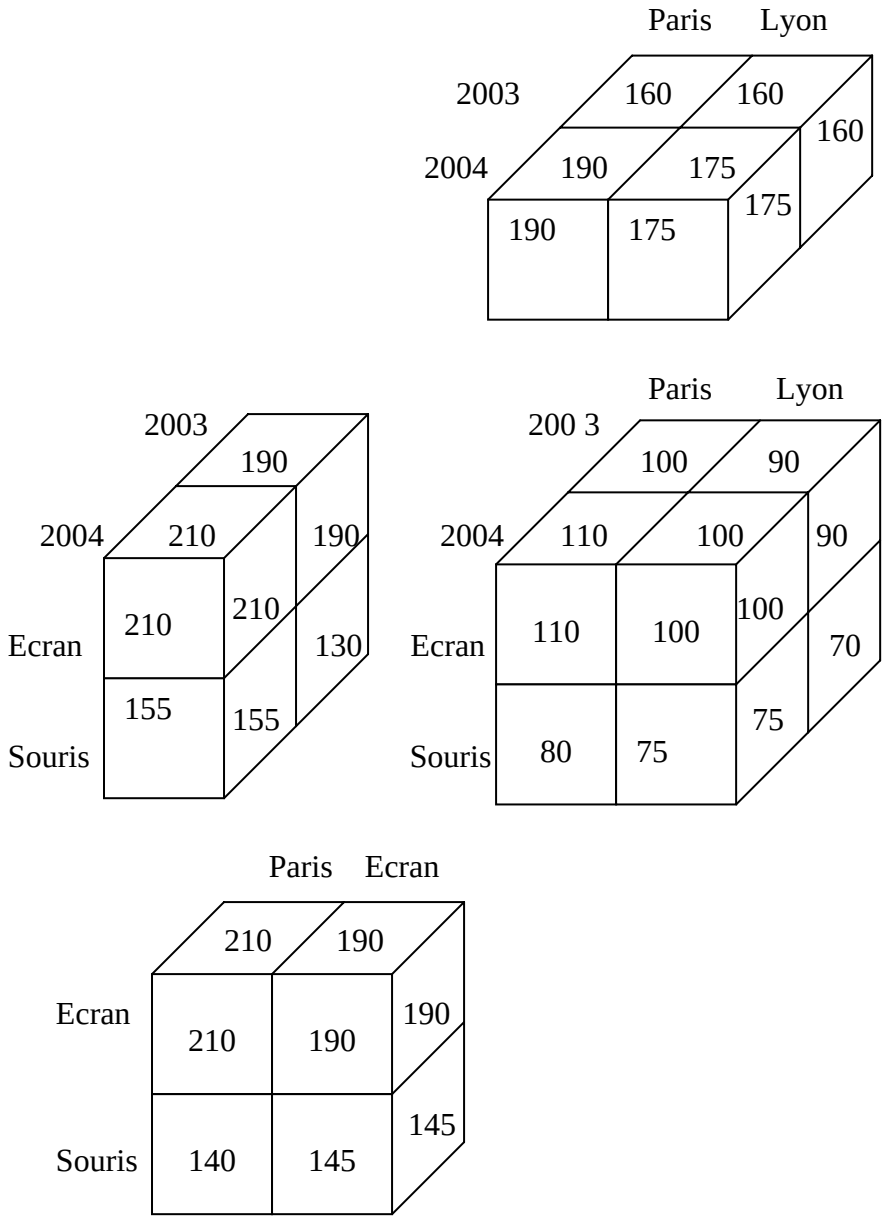
	Paris	Lyon
Ecran	100 *v1*	90 *v2*
Souris	60 *v3*	70 *v4*

2003

	Paris	Lyon
Ecran	110 *v5*	100 *v6*
Souris	80 *v7*	75 *v8*

2004

Figure 32-1 : cube initial



En TAX, cette manipulation cubique se traduit par une union des 12 sous-arbres résultants des trois forages vers le haut avec les 8 sous-arbres de départ.

Figure 32-2 : Trois forages vers le haut sur notre cube initial

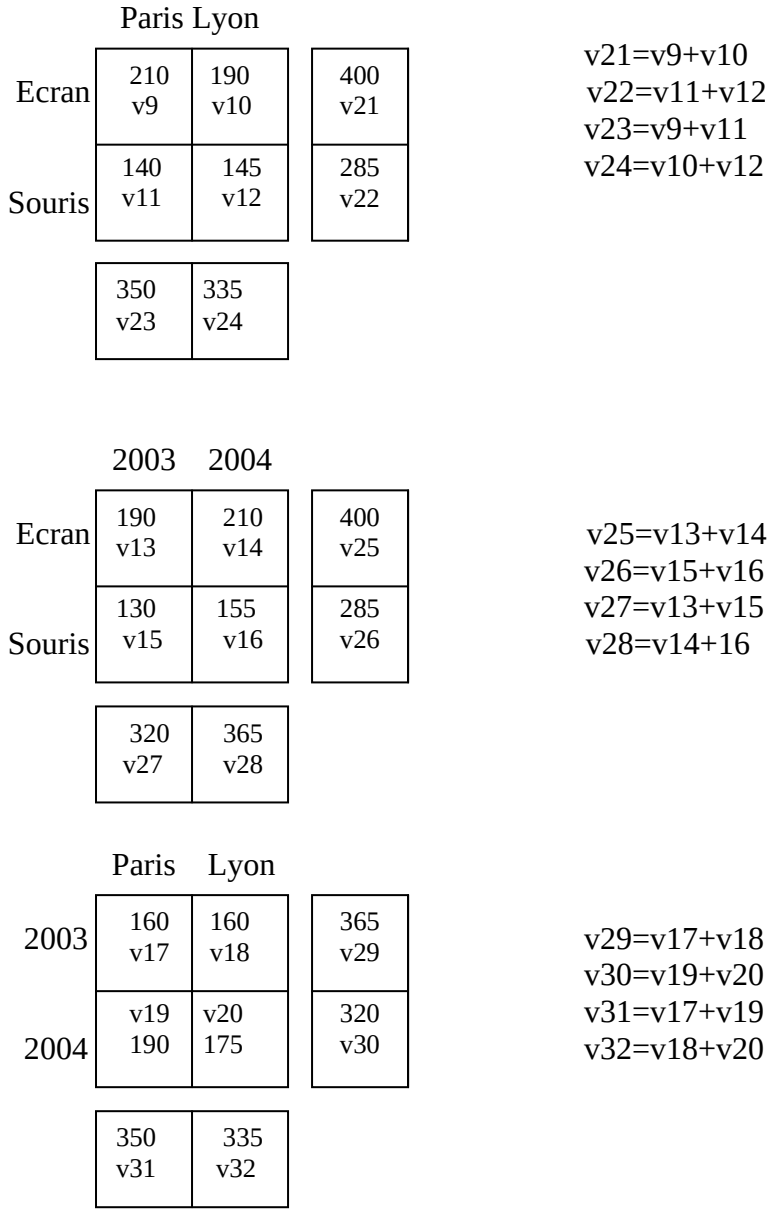


Figure 32-3 : Différentes agrégations

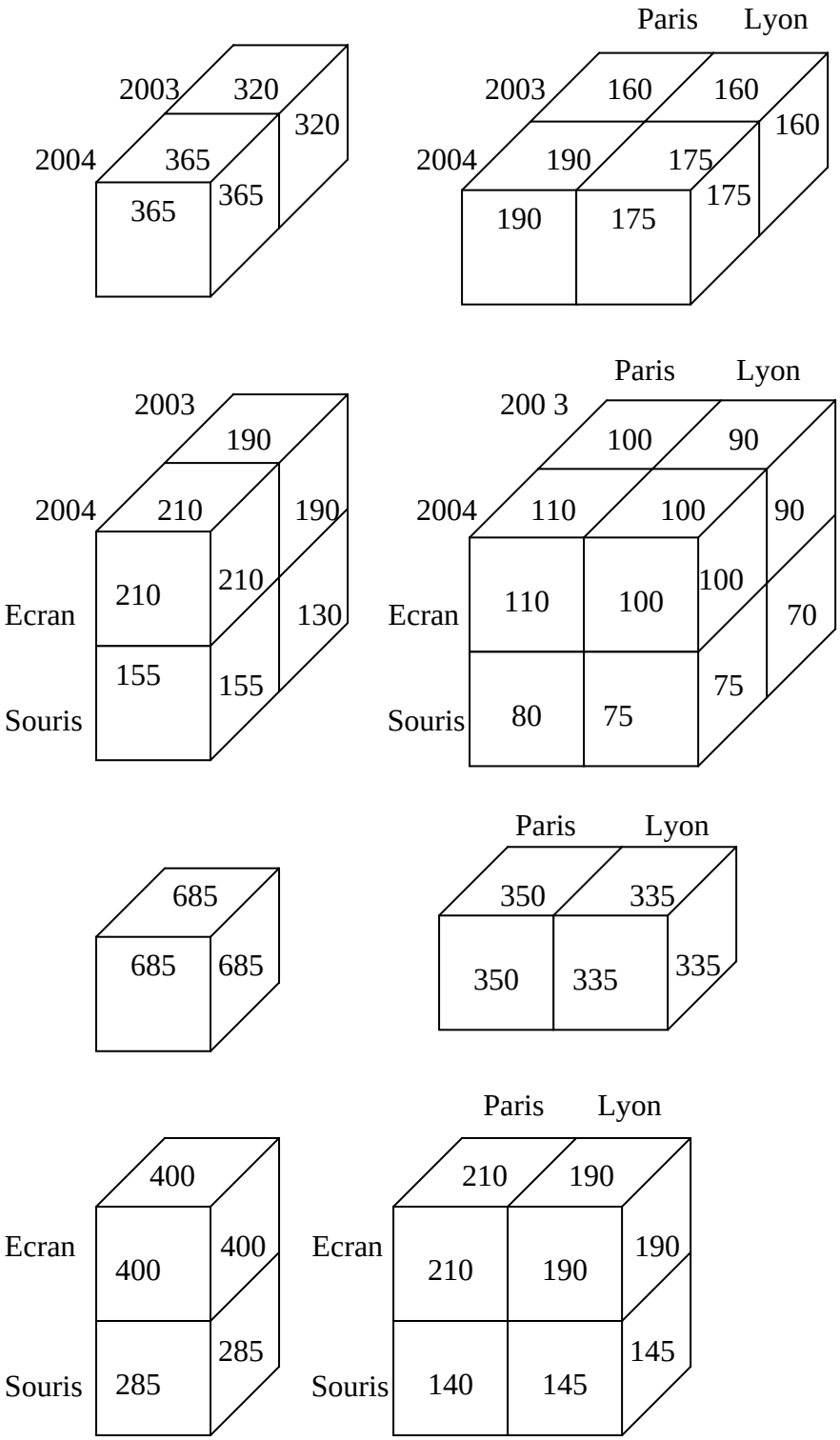


Figure 32-4 : Différentes parties du cube final

Figure 32 : Construction de cube de données avec TAX

```xml
<vente>
   <ville>Paris</ville>
   <produit>Ecran</produit>
   <annee>2004</annee>
   <prix>125</prix>
</vente>
```

```xml
<vente>
   <annee>2004</annee>
   <produit>Ecran</produit>
   <ville>Paris</ville>
   <prix>125</prix>
</vente>
```

Figure 33 : Résultat d'une rotation d'un fait XML vente avec XQuery

```
<ventes>
<vente>
    <ville>Paris</ville>
    <produit>Ecran</produit>
    <annee>2004</annee>
    <prix>125</prix>
</vente>
<vente>
    <ville>Paris</ville>
    <produit>Ecran</produit>
    <annee>2005</annee>
    <prix>100</prix>
</vente>
<vente>
    <ville>Paris</ville>
    <produit>Ecran</produit>
    <annee>2006</annee>
    <prix>160</prix>
</vente>
<vente>
    <ville>Paris</ville>
    <produit>Souris</produit>
    <annee>2004</annee>
    <prix>80</prix>
</vente>
<vente>
  <ville>Paris</ville>
  <produit>Souris</produit>
  <annee>2006</annee>
  <prix>100</prix>
</vente>
<vente>
  <ville>Paris</ville>
  <produit>Souris</produit>
  <annee>2006</annee>
  <prix>100</prix>
</vente>
<vente>
  <ville>Paris</ville>
  <produit>Clavier</produit>
  <annee>2004</annee>
  <prix>100</prix>
</vente>
<vente>
  <ville>Paris</ville>
  <produit>Clavier</produit>
  <annee>2005</annee>
  <prix>105</prix>
</vente>
<vente>
  <ville>Paris</ville>
  <produit>Clavier</produit>
  <annee>2006</annee>
  <prix>107</prix>
</vente>
</ventes>
```

Figure 34 : Résultat d'un découpage de l'entrepôt XML ventes en tranche avec XQuery

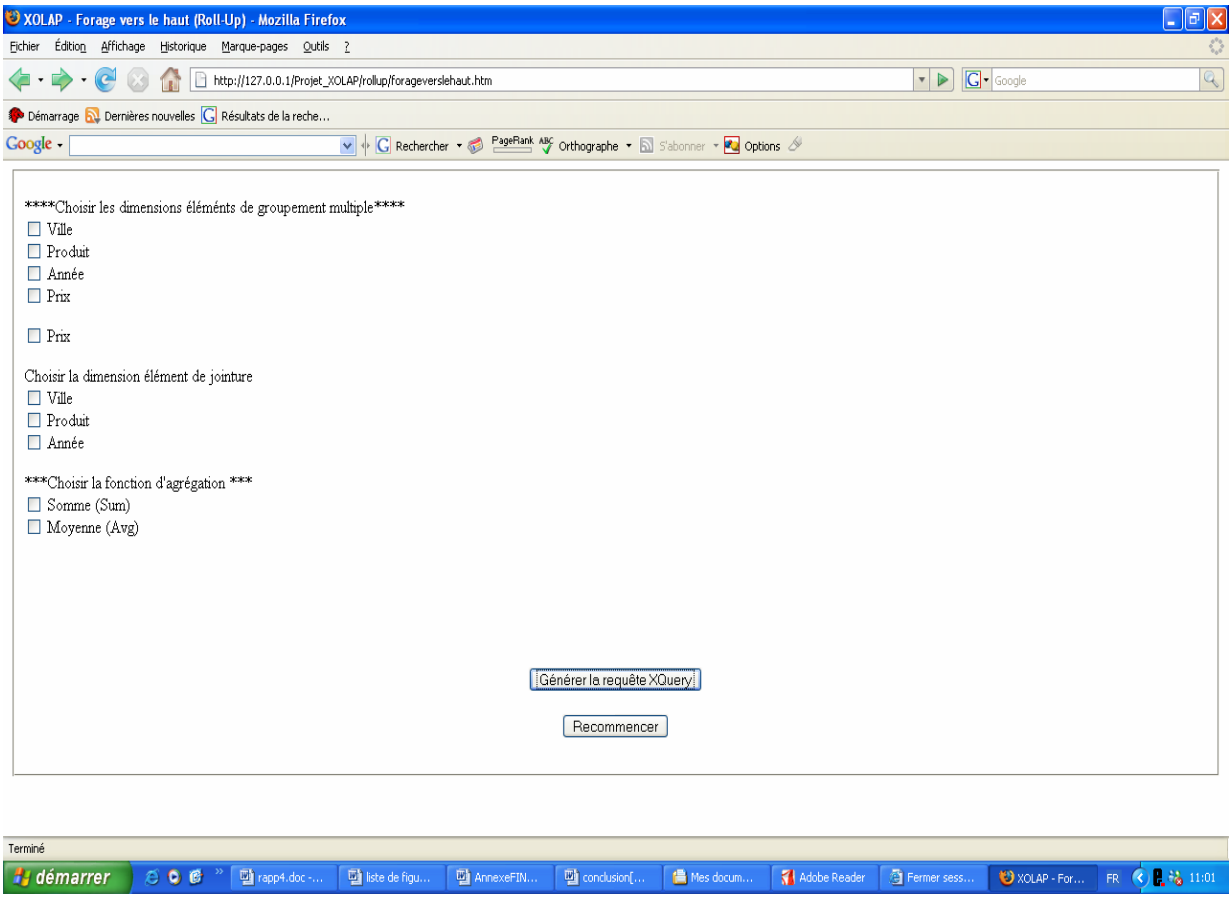


Figure 35 : Mise du forage vers le haut en pratique

```
<ventes>
<vente>
  <ville>Paris</ville>
  <pdt>Ecran</pdt>
  <temps>Temps</temps>
  <prix>385</prix>
</vente>
<vente>
  <ville>Paris</ville>
  <pdt>Souris</pdt>
  <temps>Temps</temps>
  <prix>270</prix>
</vente>
<vente>
  <ville>Paris</ville>
  <pdt>Clavier</pdt>
  <temps>Temps</temps>
  <prix>312</prix>
</vente>
<vente>
  <ville>Lyon</ville>
  <pdt>Ecran</pdt>
  <temps>Temps</temps>
  <prix>382</prix>
</vente>
<vente>
  <ville>Lyon</ville>
  <pdt>Souris</pdt>
  <temps>Temps</temps>
  <prix>293</prix>
</vente>
<vente>
  <ville>Lyon</ville>
  <pdt>Clavier</pdt>
  <temps>Temps</temps>
  <prix>593</prix>
</vente>
  <vente>
  <ville>Nice</ville>
  <pdt>Ecran</pdt>
  <temps>Temps</temps>
  <prix>615</prix>
</vente>
<vente>
  <ville>Nice</ville>
  <pdt>Souris</pdt>
  <temps>Temps</temps>
  <prix>345</prix>
</vente>
<vente>
  <ville>Nice</ville>
  <pdt>Clavier</pdt>
  <temps>Temps</temps>
  <prix>380</prix>
</vente>
</ventes>
```

Figure 36 : Résultat d'un forage vers le haut sur l'entrepôt XML ventes avec XQuery